Computing Alignments of Well-Formed Process Models using Local Search

FARBOD TAYMOURI, The University of Melbourne, Australia
JOSEP CARMONA, Universitat Politècnica de Catalunya, Spain

The alignment of observed and modeled behavior is an essential element for organizations, since it opens the door for conformance checking and enhancement of processes. The state of the art technique for computing alignments has exponential time and space complexity, hindering its applicability for medium and large instances. In this paper a novel approach is presented to tackle the challenge of computing an alignment for large problem instances that correspond to well-formed process models. Given an observed trace, first it uses a novel replay technique to find an initial candidate trace in the model. Then a local search framework is applied to try to improve the alignment until no further improvement is possible. The implementation of the presented technique reveals a magnificent reduction both in computation time and in memory usage. Moreover, although the proposed technique does not guarantee the derivation of an alignment with minimal cost, the experiments show that in practice the quality of the obtained solutions is close to optimal.

Additional Key Words and Phrases: Process Mining, Conformance Checking, Process models, Event logs.

ACM Reference Format:
Farbod Taymouri and Josep Carmona. 2020. Computing Alignments of Well-Formed Process Models using Local Search. *ACM Trans. Softw. Eng. Methodol.* 1, 1 (October 2020), 43 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Organizations are facing a digital transformation, that primarily requires an active use of the tones of data available as a result of their operation. As processes are the main focus for the management of an organization, exposing the processes to the data available helps into assessing the alignment between observed and modeled behavior. When modeled and observed behavior are aligned, then one can be sure that the reality and the models describing it agree. However, an organization may need to react in case of finding deviations between observed and modeled behavior. Conformance checking techniques [van der Aalst 2016] tackle this fundamental problem: to analytically asses the adequacy of a process model in representing the traces in an event log, extracting the deviations in case they exist. Due to the potential existence of regulations, guidelines, frauds and errors, conformance checking is becoming an essential element for an organization to prove the adherence to a desired behavior [Carmona et al. 2018].

Current conformance checking techniques strongly rely on the notion of *alignment*: given an observed trace of activities representing a process instance, to find the best model trace that resembles it [Adriansyah 2014]. Alignments are crucial for conformance checking techniques, but also open the door to improving the use of process models in organizations [Polyvyanyy et al. 2017].

Authors' addresses: Farbod Taymouri, The University of Melbourne, Doug McDonell Building, Parkville, Melbourne, 3052, Australia, farbod.taymouri@unimelb.edu.au; Josep Carmona, Universitat Politècnica de Catalunya, Jordi Girona, 1-3, Barcelona, 08034, Spain, jcarmona@cs.upc.edu.

Manuscript submitted to ACM

Due to the existence of concurrency and iteration, the behavior underlying process models can be exponential, a fact that hampers the application of the state-of-the-art technique for computing alignments, which is based on exploring the model state space. Hence, the aforementioned alignment technique can only guarantee the computation of an alignment when the process model does not exceed certain size, which in practice is often few dozen activities at most.

In practice, however, process models and event logs can be significantly bigger [4dt 2018]. When confronted to these problem instances, the available techniques will fail in providing an alignment. This is the start point for the contribution of this paper: to deploy light techniques for computing alignments for *well-formed* process models, that can be used in the large. Process models are well-formed if certain conditions on the structure of the net are satisfied. We consider that it is better to provide an alignment that is not *optimal* (i.e., that may not describe the best model run explaining the observed trace), than not providing an alignment at all. Dropping the optimality is not an objective of this work, but instead an artifact of the selection of techniques done, that are grounded on the structural theory of Petri nets, dynamic programming and local search methods. In practice, however, the results obtained are close to optimal.

Furthermore, in some situations one can live with sub-optimal solutions: For instance, when the model must be *enhanced* with the information existing in the event log (e.g., performance, decision point analysis), or when one aims to *animate* the model by replaying the log on top of it (two of the most celebrated functionalities of commercial process mining tools). In these scenarios, the information provided by a sub-optimal alignment may suffice to extend the process model.

Intuitively, the proposed technique works as follows: given a process model as a *Free-Choice Petri net* [Desel and Esparza 1995] satisfying the *workflow structure* and being *weakly sound* [van der Aalst et al. 2011], and a trace, a novel replay method is proposed. It applies the structural theory of Petri nets [Silva et al. 1998] to provide a set of transitions that should be fired, which is then used to select globally the set of transitions to replay the trace. The replay then relays in the workflow structure and the distance to the final marking (assumed to be a single place) to choose, among sets of enabled transitions, which one to fire at each reachable state in the replay. The replay is guaranteed to reach the final marking of the Petri net, thus providing a full sequence of the model. We then use a well-known technique from bioinformatics to align the two traces [Needleman and Wunsch 1970], and consequently an initial alignment is obtained. Finally, a local search technique is applied on top of the current alignment to try to improve it by merging as much as possible the deviations detected. Overall, our approach reduces the complexity to the NP class, when compared to the state-of-the-art approach.

The techniques of this paper have been implemented in a standalone tool [Taymouri 2017], and experiments done over the large problem instances publicly available nowadays reflect the capability of the technique in providing alignments in reasonable time. Remarkably, when compared to the reference technique [Adriansyah 2014] and its evolutions [van Dongen 2018], or a recent technique [Reißner et al. 2017], the results are close to optimal while there is a considerably reduction both in computation time and memory consumption, often of orders of magnitude.

The paper is organized as follows: in the next section, related work on the computation of alignments is provided. Then in Section 3 all the necessary background for understanding the techniques of this paper is introduced. Section 4 provides the framework proposed in this paper for computing alignments, which includes the three stages enumerated above. Then in Section 5 we report on the set of experiments performed and shortly present the tool developed. Section 6 concludes the paper and discusses future work.

2 RELATED WORK

The seminal work in [Adriansyah 2014] introduced the notion of alignment. For each observed trace σ in the event log, it identifies the most similar trace which can be reproduced by the model. The approach uses a depth-first search, alongside with the A^* method over the state space corresponding to the synchronous product between the model and the trace. Despite the fact that only a finite part needs to be considered on the size of the search space, it is worst-case exponential on the size of the synchronous product net. The approach is implemented in ProM [van Dongen et al. 2005], and can be considered as the state-of-the-art technique for computing alignments. A recent publication shows under which circumstances the A^* can be improved by extending the heuristic pruning the search space [van Dongen 2018].

Alternatives to the A^* have appeared very recently: in the approach presented in [de Leoni and Marrella 2017], the alignment problem is mapped as an *automated planning* instance. Unlike the A^*, the aforementioned work is only able to produce one optimal alignment (not all optimal), but it is expected to consume considerably less memory. Automata-based techniques have also appeared [Leemans et al. 2018; Reißner et al. 2017]. In particular, the technique in [Reißner et al. 2017] can compute all optimal alignments. The technique in [Reißner et al. 2017] relies on state space exploration and determinization of automata, whilst the technique in [Leemans et al. 2018] is based on computing several subsets of activities and projecting the alignment instances accordingly. A different perspective that uses *event structures* focusing in *behavioral alignments* is presented in [García-Bañuelos et al. 2018]. These alternative techniques are competitive for certain inputs, but at the same time are more sensitive to crucial aspects as problem size or degree of concurrency.

To tackle the computational challenge of computing an alignment, [Taymouri and Carmona 2016b] proposed an approach grounded on the resolution of *Integer Linear Programming* (ILP), alongside with partitioning the input observed trace σ. This technique can provide *approximate alignments*, a novel class of alignments where deviations can be explained between sets of transitions, instead of singleton pairs as in [Adriansyah 2014]. Since ILP is NP-hard, casting the problem of computing approximate alignments as the resolution of ILP models is not sufficient for alleviating the complexity of the problem. As the complexity of ILP is dominated by the number of variables and constraints, a recursive framework to compute approximate alignments that transforms the initial ILP encoding into several smaller and bounded ILP encodings is also presented. This approach reduces drastically both the memory and the CPU time required for computing approximate alignments. Although this approach is efficient both in time and memory, it is not complete, i.e., it cannot guarantee the computation of a real solution in general. A similar approach which can always guarantee a solution and heavily uses the resolution of ILP and marking equation in combination with a bounded backtracking is presented in [van Dongen et al. 2017]. The work presented in this paper takes another perspective, by only using ILP once to get an initial candidate alignment which is then improved through local search.

The technique in [Taymouri and Carmona 2016a], presents a framework to reduce a process model and the event log accordingly, with the goal to alleviate the computation of the alignments. The computed alignment, which is called *macro-alignment*, will be expanded based on the gathered information during the reduction. The approach is based on the notion of *indication*, closely related to a particular instance of the well-known notion of *synchronic distance* [Murata 1989]. The crucial observation is that transitions indicated by a given transition can be abstracted away without losing the required information for computing an alignment. Reductions done in the model can then be reflected in the trace, so that the problem instance to solve is greatly reduced. Remarkably, the approach can be integrated with other approaches of computing alignment, like the two previous ones.

A completely different line of work is by opting for a decomposition perspective [Munoz-Gama et al. 2014; van der Aalst 2013]. The proposed approach decomposes a given model to smaller parts and projects the observed trace to each corresponding part, and then computes the alignment for each part independently. This technique is very efficient, but the result is decisional (a yes/no answer on the fitness of the trace) and cannot provide a global alignment. Recently, [Verbeek and van der Aalst 2016] proposed a set of conditions for providing a global alignment from the decomposed alignments. In case the conditions are not satisfied, it produces a so-called pseudo-alignment which, as in the case of [Taymouri and Carmona 2016b], may not be executable in the net.

The approaches discussed so far provide an alignment between a model an a trace, but the work of this paper also relies on alignments between two traces seen as words, known as *sequence alignment*. In the area of process mining we are not the first proposing this: the work in [Jagadeesh Chandra Bose and van der Aalst 2010], presents the *trace alignment* approach, which given an event log, aligns the set of traces in the event log to address process model diagnostics and to explore the event log easily. As we do in this work, the refereed approach uses a dynamic programming approach for sequence alignment [Needleman and Wunsch 1970].

3 PRELIMINARIES

3.1 Sequences, Petri nets, Process Mining and Step Sequences

A *sequence* is an ordered collection of objects or alphabets in which repetitions are allowed. For example given a set of alphabets $\Sigma = \{a, b, c\}$, some sequences are ab, bac, and a, and a set of sequences is $S = \{ab, bac\}$. For two sets of sequences S_1 and S_2, the *concatenation* S_1S_2 consists of all sequences of the form uv where u is a sequence from S_1 and v is a sequence from S_2, i.e., $S_1S_2 = \{uv : u \in S_1, v \in S_2\}$. The *Kleen star* operator, *, is used to concatenate zero or more sequences from a given set. Formally, given a S of sequences, $S^* = \bigcup_{i \geq 0} S^i$. Similarly *Kleen plus* is defined as $S^+ = \bigcup_{i \geq 1} S^i$, or $S^+ = SS^*$.

A *Petri Net* [Murata 1989] is a 3-tuple $N = \langle P, T, \mathcal{F} \rangle$, where P is the set of places, T is the set of transitions, $P \cap T = \emptyset$, $\mathcal{F} : (P \times T) \cup (T \times P) \rightarrow \{0, 1\}$ is the flow relation. A *labeled Petri net (LPN)* is a 3-tuple $\langle N, \Sigma, \ell \rangle$, where N is a Petri net, Σ is an alphabet (a set of labels) and $\ell : T \rightarrow \Sigma \cup \{\tau\}$ is a *labeling function* that assigns to each transition $t \in T$ either a symbol from Σ or the empty symbol τ. The set of labeled transitions is represented by T_ℓ. A marking is an assignment of non-negative integers to places. If k is assigned to place p by marking m (denoted $m[p] = k$), we say that p is marked with k tokens. Given a node $x \in P \cup T$, its pre-set and post-set are denoted by $^\bullet x$ and $x^\bullet$ respectively and an element with the same pre-set of x is called its *sibling*. Formally, $^\bullet x = \{y | \mathcal{F}(y, x) = 1\}$, $x^\bullet = \{y | \mathcal{F}(x, y) = 1\}$. A Petri net is *Free-choice* (FC) if $\forall p_1, p_2 \in P : p_1^\bullet \cap p_2^\bullet \neq \emptyset \Rightarrow p_1^\bullet = p_2^\bullet$. Unless otherwise stated, in this paper all process models are represented by labeled FC Petri nets. For the sake of simplicity and to avoid complicated figures, labels are only shown when needed.

A transition t is *enabled* in a marking m when all places in $^\bullet t$ are marked. More formally it is denoted by $(N, m)[t\rangle$, iff $^\bullet t \leq m$. When a transition t is enabled, it can *fire* or *execute* by removing a token from each place in $^\bullet t$ and putting a token to each place in $t^\bullet$. A marking m' is *reachable* from m if there is a sequence of firings $t_1 t_2 \ldots t_n$ that transforms m into m', denoted by $m[t_1 t_2 \ldots t_n\rangle m'$. A sequence of transitions $t_1 t_2 \ldots t_n$ is a *feasible sequence* if it is firable from the initial marking m_0.

A Petri net is called *live* if no matter what marking has been reached every transitions of the model can be fired through some firing sequences. Furthermore, it is to be said *bounded* if none of the places can have infinite number of tokens for any reachable marking.

Next we define the notion of *allocation* [Desel and Esparza 1995] that is crucial for a technical result of this paper.

Definition 3.1 (Clusters). Let x be a node of a process model. The cluster x, denoted by $[x]$ is the minimal set of nodes such that:

- $x \in [x]$
- If a place p belongs to $[x]$ then $p^\bullet \in [x]$
- If a transition t belongs to $[x]$ then ${}^\bullet t \in [x]$

The above definition induces some properties. For instance, the set $\{[x] | x \in (P \cup T)\}$ is a partition of nodes of the corresponding model; another property is that in FC-nets, if an arbitrary marking enables a transition t, then it enables every transitions of $[t]$ [Desel and Esparza 1995].

Definition 3.2 (Allocations, Cyclic Allocations). Let C be a set of clusters of a model $N = \langle P, T, \mathcal{F} \rangle$, such that every cluster C contains at least one transition. An allocation is a function $\alpha : C \Rightarrow T$ such that $\forall c \in C, \alpha(c) \in c$. A transition t is said to be allocated by α if $t = \alpha(c)$ for some cluster c. Furthermore, the set of transitions allocated by α is denoted by $\alpha(C)$. An allocation α is called *cyclic* if for every cluster $c \in C$, the set $\alpha(c)^\bullet$ contains only places of C [Desel and Esparza 1995].

Workflow processes can be represented in a simple way by using Workflow Nets (WF-nets). A WF-net is a Petri net where there is a place *start* (denoting the initial state of the system) with no incoming arcs and a place *end* (denoting the final state of the system) with no outgoing arcs, and every other node is on a directed path between *start* and *end*. The transitions in a WF-net represent tasks. In conjunction with the assumption on the free-choice structure of the underlying net, this paper assumes process models are specified by *weakly sound* WF-nets [van der Aalst et al. 2011], for which every reachable marking has the option to reach the unique final marking, and there are no dead transitions.

Definition 3.3 (System Net, Full Firing Sequences). A system net is a tuple $SN = (N, m_{start}, m_{end})$, where N is a WF-net and the two last elements define the initial and final marking of the net, respectively. The set $\{\sigma \mid (N, m_{start})[\sigma\rangle(N, m_{end})\}$ denotes all the full firing sequences of SN.

Note that, a system net $SN = (N, m_{start}, m_{end})$ is live if no matter what marking has been reached, except m_{end}, every transition of the model can be fired through some firing sequences. We now turn the focus to event logs and traces:

Definition 3.4 (Trace, Event Log, Parikh vector). Given an alphabet of events $\Sigma = \{a_1, \ldots, a_n\}$, a trace is a word $\sigma \in \Sigma^*$ that represents a finite sequence of events. An *event log* $L \in \mathcal{B}(\Sigma^*)$ is a multiset of traces $|\sigma|_a$ represents the number of occurrences of a in σ. The Parikh vector of a sequence of events σ is a function $\widehat{\ }: \Sigma^* \rightarrow \mathbb{N}^n$ defined as $\widehat{\sigma} = (|\sigma|_{a_1}, \ldots, |\sigma|_{a_n})$. For simplicity, we will also represent $|\sigma|_{a_i}$ as $\widehat{\sigma}[a_i]$. The support of a Parikh vector $\widehat{\sigma}$, denoted by $\text{supp}(\widehat{\sigma})$ is the set $\{a_i | \widehat{\sigma}[a_i] > 0\}$. For a trace σ, $\sigma[1], \sigma[2], .., \sigma[k]$ denote its first, second and kth elements respectively. For two Parikh vectors $\widehat{\sigma_1}$ and $\widehat{\sigma_2}$, $\widehat{\sigma_1} \leqq \widehat{\sigma_2}$ means that each component of the former is less than or equal to each corresponding component of the later.

Alignments are strongly related to the notion of *fitness*: an alignment with no deviations describes a trace that perfectly fits the model. Consequently, the main metric in this paper to assess the adequacy of a model in describing a log is fitness [van der Aalst 2016], which is based on the reproducibility of a trace in a model:

Definition 3.5 (Fitting Trace). A trace $\sigma \in \Sigma^*$ fits $SN = (N, m_{start}, m_{end})$ if $(N, m_{start})[\sigma\rangle(N, m_{end})$.

3.2 Petri nets and Linear Algebra

Let $N = \langle P, T, \mathcal{F} \rangle$ be a Petri net with initial marking m_0. Given a feasible sequence $m_0[\sigma\rangle m$, the number of tokens for a place p in m is equal to the tokens of p in m_0 plus the tokens added by the input transitions of p in σ minus the tokens removed by the output transitions of p in σ:

$$m[p] = m_0[p] + \sum_{t \in {}^\bullet p} |\sigma|_t \; \mathcal{F}(t,p) - \sum_{t \in p^\bullet} |\sigma|_t \; \mathcal{F}(p,t)$$

The marking equations for all the places in the net can be written in the following matrix form : $m = m_0 + \mathbf{N} \cdot \widehat{\sigma}$, where $\mathbf{N} \in \mathbb{Z}^{P \times T}$ is the *incidence matrix* of the net: $\mathbf{N}[p,t] = \mathcal{F}(t,p) - \mathcal{F}(p,t)$. If a marking m is reachable from m_0, then there exists a sequence σ such that $m_0[\sigma\rangle m$, and the following system of equations has at least the solution $X = \widehat{\sigma}$

$$m = m_0 + \mathbf{N} \cdot X \tag{1}$$

If (1) is infeasible, then m is not reachable from m_0. The inverse does not hold in general: there are markings satisfying (1) which are not reachable. Those markings (and the corresponding Parikh vectors) are said to be *spurious* [Silva et al. 1998].

For well-structured Petri nets classes equation (1) characterizes reachability. The largest class is FC, *live*, bounded and *reversible* nets,i.e., the initial marking can always be reached again, [Murata 1989]. For this class, equation (1) together with a collection of sets of places (called *traps*) of the system completely characterizes reachability [Desel and Esparza 1993]. For the rest of cases, the problem of the spurious solutions can be palliated by the use of traps [Esparza and Melzer 2000], or by the addition of some special places named *cutting implicit places* [Silva et al. 1998] to the original Petri net that remove spurious solutions from the original marking equation.

Notice that the marking equation, even when characterizing faithfully the set of reachable markings, can fail to provide the right Parikh sequence between any two reachable markings. We will come back to this observation in Section 4.

3.3 Alignment of Observed Behavior

As outlined above, the fitness dimension requires an *alignment* of an observed trace and a model: events of the observed trace need to be related to elements of the model and vice versa. Such an alignment reveals how the given trace can be replayed on the process model. The classical notion of aligning an event log and process model was introduced by [Adriansyah 2014]. To achieve an alignment, we need to relate *moves* in the observed trace to *moves* in the model. It may be the case that some of the moves in the observed trace can not be mimicked by the model and vice versa. For

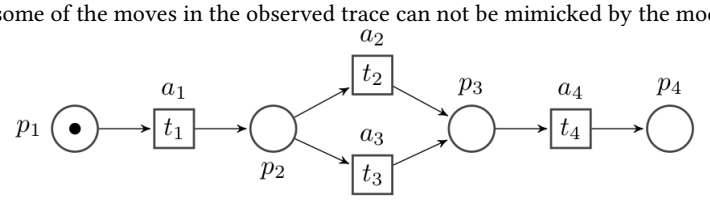


Fig. 1. Process model N_1.

instance, consider the model N_1 in Fig. 1, with the following labels, $\ell(t_1) = a_1, \ell(t_2) = a_2, \ell(t_3) = a_3$ and $\ell(t_4) = a_4$, and the trace $\sigma = a_1 a_1 a_4 a_2$; four possible alignments are:

$$\alpha_1 = \begin{array}{|c|c|c|c|c|}\hline a_1 & a_1 & \bot & a_4 & a_2 \\ \hline t_1 & \bot & t_3 & t_4 & \bot \\ \hline \end{array}, \alpha_2 = \begin{array}{|c|c|c|c|c|}\hline a_1 & a_1 & \bot & a_4 & a_2 \\ \hline \bot & t_1 & t_2 & t_4 & \bot \\ \hline \end{array}$$

$$\alpha_3 = \begin{array}{|c|c|c|c|c|}\hline a_1 & a_1 & a_4 & a_2 & \bot \\ \hline t_1 & \bot & \bot & t_2 & t_4 \\ \hline \end{array}, \alpha_4 = \begin{array}{|c|c|c|c|c|}\hline a_1 & a_1 & a_4 & a_2 & \bot \\ \hline \bot & t_1 & \bot & t_2 & t_4 \\ \hline \end{array}$$

The moves are represented in tabular form, where moves by the trace are at the top, and moves by the model are at the bottom of the table. For example the first move in α_2 is $(a_1, \bot)$ and it means that the observed trace moves a_1, while the model does not make any move. Formally, an alignment is defined as follows:

Definition 3.6 (Alignment). Given a labeled Petri net N and an alphabet of events Σ, Let A_M and A_L be the alphabet of transitions in the model and events in the log, respectively, and $\bot$ denote the empty set, then:

- (X, Y) is a *synchronous move* if $X \in A_L$, $Y \in A_M$ and $X = \ell(Y)$
- (X, Y) is a *move in log* if $X \in A_L$ and $Y = \bot$.
- (X, Y) is a *move in model* if $X = \bot$ and $Y \in A_M$.
- (X, Y) is an *illegal move*, otherwise.

The set of all *legal* moves is denoted as A_{LM} and given an alignment $\alpha \in A^*_{LM}$, the projection of the first element (ignoring $\bot$), $\alpha \downarrow_{A_L}$, results in the observed trace σ, and projecting the second element (ignoring $\bot$), $\alpha \downarrow_{A_M}$, results in the model trace.

For the previous example, $\alpha_1 \downarrow_{A_M} = t_1 t_3 t_4$ and $\alpha_1 \downarrow_{A_L} = a_1 a_1 a_4 a_2$.

Cost can be associated to alignments, with asynchronous moves having greater cost than synchronous ones [Adriansyah 2014]. Once cost are defined, an alignment with optimal cost is preferred.

Definition 3.7 (Cost of an Alignment). We define the cost function $\lambda : A_{LM} \rightarrow \mathbb{N}$, as follows:

$$\forall (X, Y) \in A_{LM} \qquad \lambda((X, Y)) = \begin{cases} \delta_S & \text{If} \quad X = \ell(Y) \\ \delta_L & \text{If} \quad Y = \bot \quad \text{And} \quad 0 \leq \delta_S < \delta_L, \delta_M \\ \delta_M & \text{If} \quad X = \bot \end{cases} \tag{2}$$

δ_S is the cost associated to a synchronous move, and δ_M and δ_L are the costs for asynchronous (move-in-model and move-in-log) moves, respectively. Therefore the cost of an alignment can be summed over costs of its moves, i.e., $\lambda(\alpha) = \Sigma_{(X,Y)\in\alpha} \lambda((X, Y))$.

Given an alignment computed over an observed trace σ, *fitness* (see Def. 3.5) can be defined as the ratio given by the number of events in σ which can be mimicked by the model, i.e., synchronous moves, to the total number of moves in α. The fitness value, is the normalized associated cost, that is a quantity between 0 and 1, thus, the closer fitness value is to 1, the more similar is the model trace to the given observed trace. Formally:

Definition 3.8 (Cost Based Fitness Metric). For a given alignment α, the fitness value, π_α, is defined as follows:

$$\pi_\alpha = 1 - \frac{(\Sigma_{(X,\bot)\in\alpha} \delta_L + \Sigma_{(\bot,Y)\in\alpha} \delta_M)}{(\Sigma_{(X,Y)\in\alpha} \delta_S + \Sigma_{(X,\bot)\in\alpha} \delta_L + \Sigma_{(\bot,Y)\in\alpha} \delta_M)} \tag{3}$$

Def. 3.8 is the ratio of the total cost of synchronous moves to the whole cost of the alignment, i.e., synchronous and asynchronous moves. It is apparent from Eq. (3) that when there are no synchronous moves, i.e., $\Sigma_{(X,Y)\in\alpha} \delta_S = 0$,

$\pi_\alpha = 0$. On the other hand, if there are no deviations, i.e., $\Sigma_{(X,\perp)\in\alpha}\delta_L = 0$ and $\Sigma_{(\perp,Y)\in\alpha}\delta_M = 0$, then $\pi_\alpha = 1$. As an example given costs $\delta_S = 1, \delta_L = 2$ and $\delta_M = 2$, $\pi_{\alpha_3} = 1 - \frac{2\times2+1\times2}{2\times1+2\times2+1\times2} = 0.25$.

It is worth noting that regardless of the aforementioned example, in general, any non-negative cost values as defined in Eq. 2 can be considered. However, in this paper for the sake of simplicity we assume $\delta_s = 1$ for all synchronous moves, and $\delta_L = \delta_M = 2$ for all asynchronous moves.

4 LOCAL SEARCH COMPUTATION OF ALIGNMENTS

4.1 The Overall Perspective

Fig. 2 represents the overall proposed framework. The details of each part will be presented in upcoming sections. Short descriptions of each part are presented here:

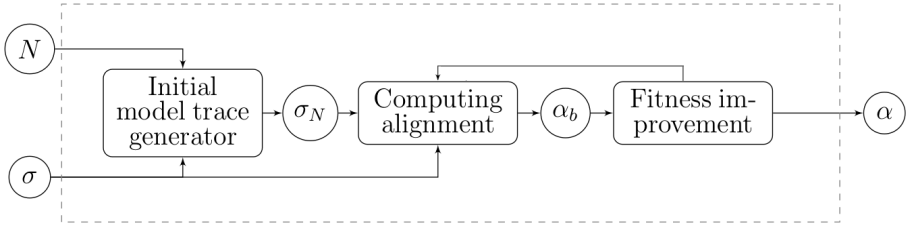


Fig. 2. General idea for local search computation of alignments.

- *Initial modeled trace generator.* Given a model, N, and an observed trace σ, in this stage an initial modeled trace σ_N is computed. Obviously, given a cost function, the closer σ_N is to σ, the better. For solving this problem, some possibilities exist, based on *replay* [Rozinat and van der Aalst 2008; vanden Broucke et al. 2014]. In this work we will use a novel technique that incorporates the marking equation, i.e., Eq. (1), in order to attain the maximum similarity to the observed trace in terms of number of events fired, a term that we denote *Parikh similarity*. Then, since the set of firings is computed as the solution of the marking equation, a replay technique that considers the distances in the graph underlying the Petri net is used to obtain the real sequence. This approach has interesting features that makes it to avoid backtracking in the search for a model trace. See Alg. 1 lines 2-4.
- *Computing an Alignment.* Computing an alignment between the observed and modeled traces can be done through a well-known dynamic programming approach inspired from the bioinformatics field. The result is the initial base alignment, α_b. See Alg. 1 lines 6-8.
- *Fitness improvement.* The initial modeled trace, σ_N, upon which α_b is based, is not guaranteed in general to be the best possible alignment. To improve the fitness value of the initial alignment, a local search strategy is applied to reduce the number of asynchronous moves in α_b, while preserving the executable property of the modeled trace. See Alg. 1 line 10.

This way, fitness improvement can be done iteratively, i.e., recomputing the current alignment α_b based on the previous improved alignment, until no more improvements can be made. This explains the arc-back from the fitness improvement part to the computing alignment part in Fig. 2.

Alg. 1 presents the general template for the framework proposed in this paper. Notice that particular instantiations can be done in some parts of the algorithm, so that a personalized version can be obtained by selecting each important piece. For instance, by choosing one or other replay technique, a different version will be derived. Next sections will

provide further details on each one of the stages.

Algorithm 1 Overall Framework

Input: Global Variables $SN(N, m_{start}, m_{end})$, σ ▷ Inputs are the system net, observed trace
#Initial modeled trace generator
  $\hat{\sigma_P} \leftarrow$ Result of **Eq. 4** ▷ Computing Parikh vector, $\hat{\sigma_P}$
  $\sigma_N \leftarrow$ **Execute** $(\hat{\sigma_P})$ ▷ Replaying $\hat{\sigma_P}$ to get model trace σ_N
while (Fitness can be improved) **do**
  #Computing alignment
    $M, S \leftarrow$ **Align** (σ, σ_N) ▷ Aligning σ, σ_N to get score and source matrices M, S
    $\alpha_b \leftarrow$ **Traceback** (S) ▷ Using source matrix S, to traceback and obtain α_b
  #Fitness improvement
    $\alpha \leftarrow$ **Alignment-Reordering** (α_b) ▷ Reordering α_b to improve π_{α_b}(fitness value)
  $\alpha_b \leftarrow \alpha$
  $\sigma_N \leftarrow \alpha_b \downarrow_{A_M}$
end while
Return α_b

4.2 Initial model trace generator

Given an observed trace σ the objective of this part is to generate an executable modeled sequence σ_N, whose Parikh vector has the maximum similarity to $\widehat{\sigma}$. In this paper we propose a new approach for this problem, which can be performed in two stages. In case of Free-choice Petri nets and weakly sound models, formal guarantees will be provided that ensure the derivation of a model trace.

The first stage, which is called ILP similarity, is originally presented in [Taymouri and Carmona 2016b]. By using equation (1), it relies on the resolution of an ILP, which provides a Parikh vector $\widehat{\sigma_P}$ whose elements are as much similar as possible to the elements of $\widehat{\sigma}$. Given $\widehat{\sigma_P}$, the second stage presents a novel technique to replay the computed Parikh vector to get an executable sequence σ_N, which excludes replaying spurious elements of $\widehat{\sigma_P}$. It will be proved that given the solution of the first stage, the second stage is complete, i.e., it is able to find a solution.

ILP for Similarity: Seeking for an Optimal Parikh Vector. This stage will be centered on the marking equation of the input Petri net. Let $J = \Sigma \cap \text{supp}(\widehat{\sigma})$, i.e., the labels that appeared in the observed trace, the following ILP model computes a solution X (representing a Parikh vector of a sequence σ_P) that is as similar as possible with respect to the firing of the activities appearing in the observed trace:

$$\text{Maximize } \Big(\sum_{\ell(t)\in J} X[t] - \delta \times \sum_{\ell(t)\notin J} X[t] + 0 \times \sum_{\ell(t)=\tau} X[t] \Big),$$

$$\text{Subject to:}$$

$$m_{end} = m_{start} + \mathbf{N}.X \tag{4}$$

$$\forall t \in X, \forall a \in \widehat{\sigma} \quad \text{If} \quad \ell(t) \in J \quad \text{and} \quad \ell(t) = a : \quad \widehat{\sigma}[a] = \sum_{\ell(t)=a} (X[t] + X^s[t])$$

$$X, X^s \in \{\mathbb{N}, 0\}^{|J|}$$

δ in the objective function is a user defined value with $\delta \geq 1$, which penalizes transitions of the model which do not have any labels in J. The larger is the value of δ, the greater penalty the elements not in J do receive. Also note that

invisible transitions of the model, i.e., $\ell(t) = \tau$, receive 0 cost. Hence, model (4) searches for a vector X that is both a solution to the marking equation, and maximizes the similarity with respect to $\widehat{\sigma}$. Notice that the ILP problem has an additional set of variables X^s, which represent the *slack variables* needed when a solution for a given transition cannot equal the observed number of firings in the trace. By maximizing elements of X in J and minimizing those not in J, solutions to (4) clearly try to both assign zeros as much as possible to the X^s variables on the one side, and on the other side, try to do not fire the X variables not in J (i.e., activities not appearing in σ). Also, if for an arbitrary event in the observed trace there are some transitions of the model with the same label, then the number of firings for that event is equal to sum over all those transitions with that label.

An optimal solution X to (4), denoted by $\widehat{\sigma_P}$, represents the required transitions of the model and their number of occurrences which must be fired from the initial marking, m_{start}, to reach the final marking, m_{end}. Elements of $\widehat{\sigma_P}$ have the maximum similarity with respect to $\widehat{\sigma}$, a phenomena that we denote *Parikh similarity*.

For example, consider the model N_2 depicted in Fig. 3 and the observed trace $\sigma = a_1a_2a_4a_5a_7a_4a_3a_6$, where $\widehat{\sigma}$ is depicted in Fig. 5 (b) and $J = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$. Duplicate transitions are highlighted (t_2 and t_3, denoting the same label a_4). Setting $\delta = 1$ in Eq. (4) for these model and trace (described in detail in Fig. 4) results in the Parikh vector, X, which is called $\widehat{\sigma_P}$. X (and the slack variables X^s) is depicted in Fig. 5 (a).

Notice that a_4 occurred twice in σ, but based on N_2 only one of them is executed, since otherwise a token would be missed at place p_1. Transitions t_2 and t_3 with the label a_4 are eligible to be fired accordingly, hence in Fig. 4, the following constraint is proposed, i.e., $X[t_2] + X^s[t_2] + X[t_3] + X^s[t_3] = 2$. Furthermore,, the occurrence of a_2 constitutes the constraint $X[t_8] + X^s[t_8] = 1$. Finally, based on the just mentioned constraint, Eq. (4) assigns $X[t_8] = 1$ and $X[t_2] = 1$ to maximize the objective function, increasing the similarity between elements of X and $\widehat{\sigma}$. Hence in Fig. 5 (a) it assigns $X[t_3] = 0$ and $X^s[t_2] = 1$ or $X^s[t_3] = 1$ and $X^s[t_2] = 0$ to make the corresponding constraint valid[1]. Note that a_7 occurred once in the observed trace, i.e., $\widehat{\sigma}[a_7] = 1$ and therefore it suggests the constraint $X[t_7] + X^s[t_7] = 1$, but t_7 will not be fired because otherwise the solution is infeasible[2], hence $X[t_7] = 0$ and $X^s[t_7] = 1$. Also, notice that no event corresponding to transition t_{10} of the model occurred in σ, hence in the objective function this transition was penalized. Transition t_9 ($\ell(t_9) = \tau$), is a silent transition hence in the objective function it receives 0 cost. Finally, Fig. 5 (b) describes both $\ell(X)$, the computed Parikh vector with element labels, and $\widehat{\sigma}$, so that the reader can compare the similarity between elements of the two vectors.

[1]Notice that the choice of assignment for X^s variables does not matter for the present approach.
[2]If t_7 is fired, then reaching the final marking based on the remaining constraints is infeasible because in that case t_5 and t_4 must be fired twice, but this contradicts the constraints $X[t_5] + X^s[t_5] = 1$ and $X[t_4] + X^s[t_4] = 1$.

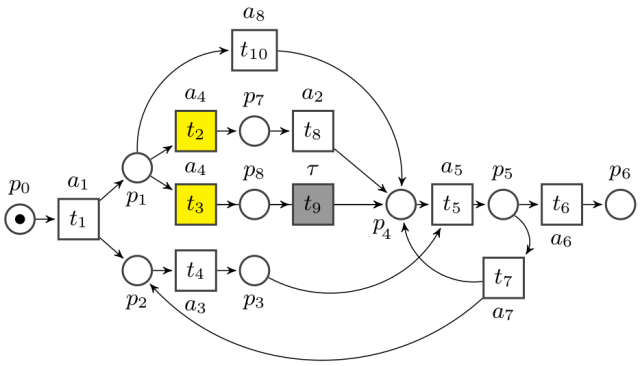


Fig. 3. Process model N_2.

Maximize $(X[t_1] + X[t_2] + X[t_3] + X[t_4]$
$+ X[t_5] + X[t_6] + X[t_7] + X[t_8] - 1 \times X[t_{10}] + 0 \times X[t_9])$

Subject to:

$$\underbrace{\begin{pmatrix}0\\0\\0\\0\\0\\0\\1\\0\\0\end{pmatrix}}_{m_{end}} = \underbrace{\begin{pmatrix}1\\0\\0\\0\\0\\0\\0\\0\\0\end{pmatrix}}_{m_{start}} + \underbrace{\begin{array}{c} \\ P_0\\P_1\\P_2\\P_3\\P_4\\P_5\\P_6\\P_7\\P_8\end{array}\begin{array}{c}\begin{array}{cccccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 & t_{10}\end{array}\\\begin{pmatrix}-1&0&0&0&0&0&0&0&0&0\\1&-1&-1&0&0&0&0&0&0&-1\\1&0&0&-1&0&0&1&0&0&0\\0&0&0&1&-1&0&0&0&0&0\\0&0&1&0&-1&0&1&1&1&1\\0&0&0&0&1&-1&-1&0&0&0\\0&0&0&0&0&1&0&0&0&0\\0&1&0&0&0&0&0&-1&0&0\\0&0&0&0&0&0&0&0&-1&0\end{pmatrix}\end{array}}_{\mathbf{N}} \times \underbrace{\begin{pmatrix}t_1\\t_2\\t_3\\t_4\\t_5\\t_6\\t_7\\t_8\\t_9\\t_{10}\end{pmatrix}}_{X}$$

Constraints

$$\left.\begin{array}{l} X[t_1] + X^s[t_1] = \hat{\sigma}[a_1],\\ X[t_2] + X^s[t_2] + X[t_3] + X^s[t_3] = \hat{\sigma}[a_4],\\ X[t_4] + X^s[t_4] = \hat{\sigma}[a_3],\\ X[t_5] + X^s[t_5] = \hat{\sigma}[a_5],\\ X[t_6] + X^s[t_6] = \hat{\sigma}[a_6],\\ X[t_7] + X^s[t_7] = \hat{\sigma}[a_7],\\ X[t_8] + X^s[t_8] = \hat{\sigma}[a_2],\\ X \geq 0, X^s \geq 0\\ m_{start} \geq 0, m_{end} \geq 0 \end{array}\right\} \quad \text{or} \quad \left.\begin{array}{l} X[t_1] + X^s[t_1] = 1,\\ X[t_2] + X^s[t_2] + X[t_3] + X^s[t_3] = 2,\\ X[t_4] + X^s[t_4] = 1,\\ X[t_5] + X^s[t_5] = 1,\\ X[t_6] + X^s[t_6] = 1,\\ X[t_7] + X^s[t_7] = 1,\\ X[t_8] + X^s[t_8] = 1,\\ X \geq 0, X^s \geq 0\\ m_{start} \geq 0, m_{end} \geq 0 \end{array}\right\} \text{Constraints}$$

Fig. 4. ILP formulation for model N_2.

$$X = \begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \\ t_9 \\ t_{10} \end{matrix}\begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \qquad X^s = \begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \end{matrix}\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

(a)

$$\ell(X) = \begin{matrix} a_1 \\ a_4 \\ a_4 \\ a_3 \\ a_5 \\ a_6 \\ a_7 \\ a_2 \\ \tau \\ a_8 \end{matrix}\begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \qquad \hat{\sigma} = \begin{matrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{matrix}\begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

(b)

Fig. 5. (a) ILP solution for the model N_2., (b) X with labels and $\widehat{\sigma}$.

Replay The Parikh Vector: Computing An Executable Model Trace. This stage consists of executing the computed Parikh vector $\widehat{\sigma_P}$, in order to get a feasible sequence σ_N. As commented before, this step assumes a model specified over a FC, weakly sound Petri net[3]. The benefits of getting a sequence, from the computed Parikh vector $\widehat{\sigma_P}$ are twofold. First, for computing the initial alignment α_b, both observed and modeled traces are needed whereas at this point there is no information about the order of elements in $\widehat{\sigma_P}$. Second, because the computed Parikh vector is the resolution of an ILP instance based on the marking equation, i.e. Eq. (1), it may contain some spurious elements (see section 3.2). Therefore to fix this problem, i.e., getting a real solution which can be executed, $\widehat{\sigma_P}$ is replayed from the initial marking to the final marking. Obviously, there is not a unique way to execute a given Parikh vector.

The most simple way of executing a given Parikh vector $\widehat{\sigma_P}$ is based on the idea of using a tree based search approach or "generate and test", which could be done with the use of a depth-first search (DFS) mixed with a backtracking approach. This approach starts at the initial marking and fires transitions in $\widehat{\sigma_P}$ as much as possible, and backtracks whenever a deadlock situation is encountered. In the worst case, this approach may require to explore the full set of reachable markings in order to find a solution.

The approach proposed in this paper prevents the backtracking mechanism. In our technique, a transition of the model is selected to be fired depending on its graph distance to the final marking (i.e., to the place corresponding to the final marking): intuitively, a transition with the farthest distance to the final marking is on priority to be fired. To achieve this goal, the Floyd–Warshall algorithm, which computes the shortest path between two arbitrary nodes in a graph, is used as the main criterion. The computed shortest paths are represented by the distance matrix D, which is computed only once for the graph underlying the WF-net. Avoiding backtracking is the advantage of this approach, which comes at the expense of getting a modeled trace that is not guaranteed to resemble optimally the observed trace, thus potentially lowering the quality of the corresponding alignment. In practice however, the sequences obtained by this approach are rather useful in our setting. Definition 4.1 states the corresponding firing policy based on the mentioned criterion. It provides only the necessary criterion for a transition to be on priority of firing, whereas the sufficient condition will be discussed shortly afterwards.

Definition 4.1 (Firing policy). Given Parikh vector $\widehat{\sigma_P}$ and system net $SN = (N, m_{start}, m_{end})$ with $m_{end} = \{p_e\}$, where p_e is the end place in SN; Let $T_{m_i} = \{t \in T | {}^\bullet t \leq m_i\}$, i.e., enabled transitions in marking m_i, and $T_{m_i} \cap supp(\widehat{\sigma_P}) =$

[3]In spite of this assumption, in practice it performs well for more general classes, including unstructured Petri nets. This is illustrated in the experimental results provided in this paper.

$\{t_1, t_2, ..., t_n\}$, i.e., enabled transitions that belong to $\widehat{\sigma_P}$ in marking m_i. If $D(t_k, p_e)$ represents the minimum distance of t_k to p_e then it is on priority to be fired if $\forall i \neq k : t_i \in T_{m_i} \cap supp(\widehat{\sigma_P})$, $D(t_k, p_e) > D(t_i, p_e)$.

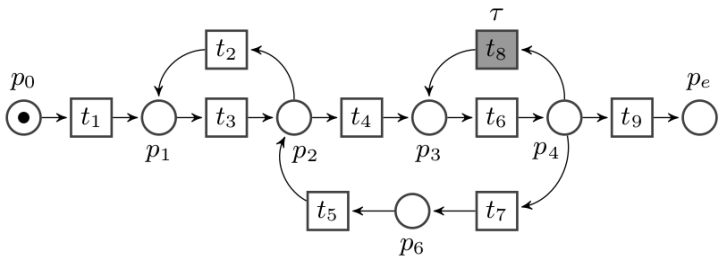


Fig. 6. Process model N_3.

Notice that in Def. 4.1, there may be more than one transition in priority to be fired, in which case any of them can be fired according to the definition. Replaying $\widehat{\sigma_P}$ based on Def. 4.1 continues until the final marking is reached (the proof of reaching the final marking will be presented later). After firing t_k, the marking of the model and transitions remaining in $\widehat{\sigma_P}$ are updated accordingly. Also notice that labels of transitions, i.e., duplicate or invisible labels, do not pose any problem with respect to the firing policy in Def. 4.1, since transitions are to be fired based on the computed Parikh vector $\widehat{\sigma_P}$ and not on $\ell(\widehat{\sigma_P})$.

Consider the model N_3 and computed Parikh vector in Figures 6 and 7 (a), respectively. The initial and final marking of the model are $m_{start}[p_0] = 1$, $m_{end}[p_e] = 1$ respectively. DFS starts at t_1, and without loss of generality let us assume that given a set of enabled transitions the one with larger subscript is chosen first, i.e, between t_8 and t_9, the latter will be expanded (fired) first. Executing $\widehat{\sigma_P}$ via backtracking by DFS, is represented in Fig. 7 (b) and red colors represent where the algorithm has to backtrack. On the other hand executing $\widehat{\sigma_P}$ using Def. 4.1 does not need any backtracking. See Fig. 7 (d), distances are represented on the top of each transition. The gray colored transitions represent those which are enabled at the corresponding marking, i.e. $T_{m_i} \cap supp(\widehat{\sigma_P})$, but they are not on priority to be fired due to Def. 4.1. For example if $m[p_2] = 1$, then both t_2 with $D(t_2, p_e) = 9$ and t_4 with $D(t_4, p_e) = 5$ are enabled, but the former will be fired since it is farther than the later from the final marking, or when $m[p_4] = 1$ then t_8 with $D(t_8, p_e) = 5$, t_9 with $D(t_9, p_e) = 1$ and t_7 with $D(t_7, p_e) = 9$ are enabled, where the later is farther with respect to others to the final marking, therefore it will be fired. Finally the computed modeled traces σ_N by two approaches are represented in Fig. 7 (b) and (d), respectively.

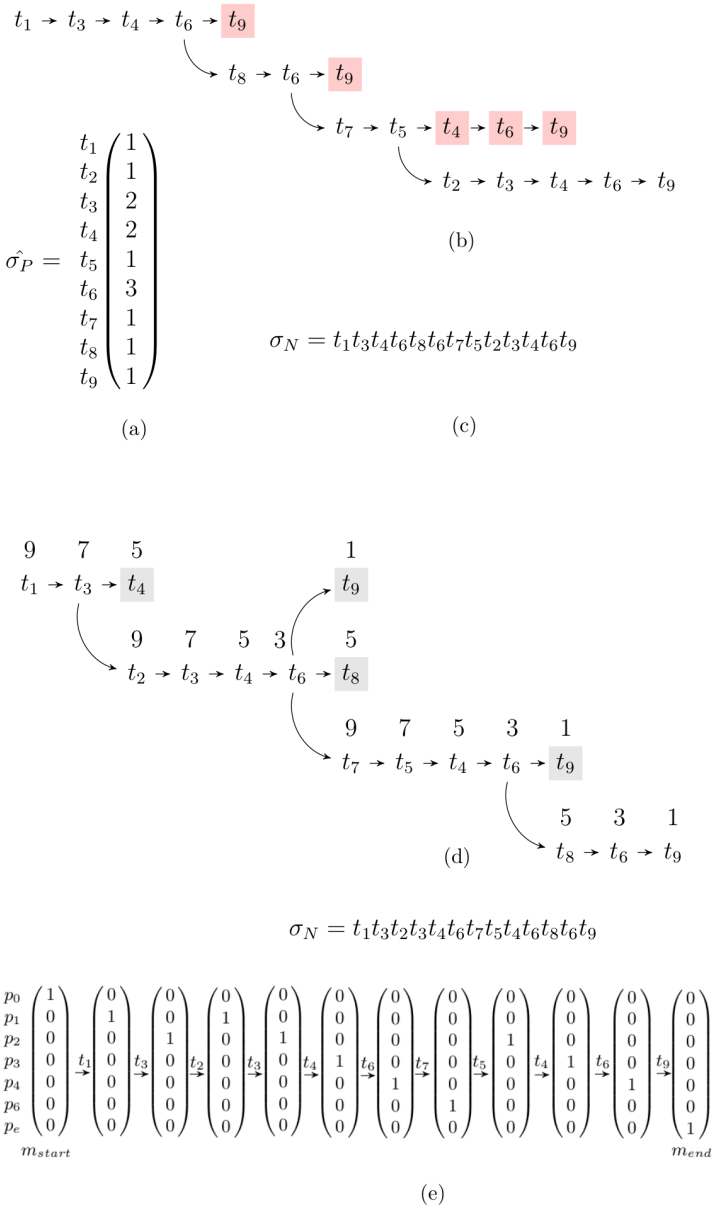


Fig. 7. (a) Parikh vector, (b) Executing $\widehat{\sigma_P}$ using DFS, (c) The modeled trace computed by DFS, (d) Executing $\widehat{\sigma_P}$ using the proposed approach, (e) The modeled trace computed by the proposed approach

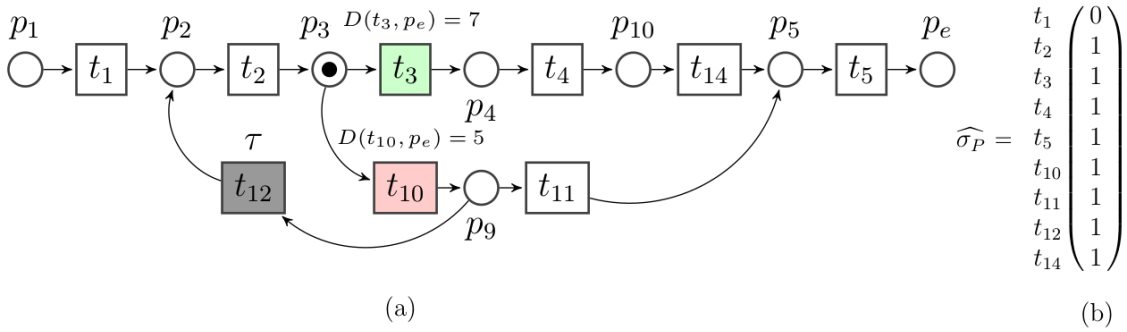


Fig. 8. (a) Parikh vector, (b) Process model N_4.

As mentioned already, replaying $\widehat{\sigma_P}$ based on Def. 4.1 , i.e., granting priority of firing to the farthest transition with respect to p_e avoids backtracking, but in some situations causes to miss some other transitions which are supposed to be fired based on $\widehat{\sigma_P}$. Hence, being in the farthest distance with respect to p_e, albeit necessary, is not a sufficient criterion of being on priority of firing for the technique of this paper.

For example consider the model N_4 in Fig. 8 (a), after firing t_1 and t_2 the corresponding Parikh vector is shown in Fig, 8 (b). One sees that both t_3 and t_{10} are enabled in $supp(\widehat{\sigma_P})$. Based on the given marking of the model, t_3 which is the farthest transition with respect to p_e, has the priority of firing, while doing so causes t_{10} to be missed in the replay, although it is supposed to be fired according to $\widehat{\sigma_P}$. To avoid such problems, a transition with priority of firing must be fired *legitimately* according to Def. 4.2 as follows:

Definition 4.2 (Legitimacy). Assume t_k is defined according to Def. 4.1, and let $T_S = \{t_i | t_i \in T_{m_i} \cap supp(\widehat{\sigma_P}), {}^\bullet t_i = {}^\bullet t_k\}$, i.e., T_S represents the enabled siblings of t_k which are in $supp(\widehat{\sigma_P})$ as well, then t_k will be fired legitimately if and only if, $\exists t_j \in T_S$ and $\exists c \in \mathbb{N}$, where $D(t_k, t_j) = c$.

Def. 4.2 informally states that, the priority of firing is granted to the farthest transition t_k if and only if, at least one of its siblings in T_S would be structurally accessible after it is fired. Notice that, as it happens with Def. 4.1, there may be more than one transition legitimated to be fired according to Def. 4.2, which would require to pick any of them for firing. Def. 4.1 and 4.2 provide the necessary and sufficient criterion for a transition to be fired on priority which guarantees no elements in $T_{m_i} \cap supp(\widehat{\sigma_P})$ are missed, i.e. all of them will be fired without backtracking. It can be extended to all encountered reachable markings and corresponding enabled transitions of $\widehat{\sigma_P}$ given that they are not spurious (spurious elements, by definition, will not get enabled).

Theorem 4.3 (Completeness given legitimate firing). *Given the context provided on a transition t_k by Def. 4.2, all elements of $T_{m_i} \cap supp(\widehat{\sigma_P})$ are fired without backtracking if and only if t_k fired legitimately.*

Prior to prove Theorem 4.3, first it must be established that for a reachable marking m_i, all the corresponding enabled transitions, i.e. $T_{m_i} \cap supp(\widehat{\sigma_P})$, are able to be fired in some order. Formally, it is presented in the following Lemma. This Lemma can be extended to all reachable markings from m_{start} to m_{end}.

Lemma 4.4. *Let Parikh vector $\widehat{\sigma_P}$, system net $SN = (N, m_{start}, m_{end})$ and an arbitrary marking $m_i \neq m_{end}$ that is reachable from m_{start}. $\forall t_j \in T_{m_i} \cap supp(\widehat{\sigma_P})$, there exists at least one sequence of transitions σ_O such that $supp(\widehat{\sigma_O}) \subseteq supp(\widehat{\sigma_P})$, $\widehat{\sigma_O} \leqq \widehat{\sigma_P}$ and $\widehat{\sigma_O}[t_j] = \widehat{\sigma_P}[t_j]$.*

The proof of Lemma 4.4 is presented in Appendix A.1. It must be emphasized that Lemma 4.4 declares, given marking m_i, for all enabled transitions of $supp(\widehat{\sigma_P})$, there exists at least one firing sequence according to their occurrences. It does not claim that all elements in $\widehat{\sigma_P}$ are able to be fired in some order entirely. Furthermore,, the following consequence can be obtained from Lemma 4.4.

Lemma 4.5. *For an arbitrary marking m_i, let $t_k \in T_{m_i} \cap supp(\widehat{\sigma_P})$ and let without loss of generality $\widehat{\sigma_P}[t_k] = n_k = 1$[4]. Then by virtue of Lemma 4.4, there exists a sequence of firing, T_M, where $supp(\widehat{\sigma_{T_M}}) \subseteq supp(\widehat{\sigma_P})$, with $\widehat{\sigma_{T_M}} \leqq \widehat{\sigma_P}$ such that the following holds, $m_y = m_i + \mathbf{N} \cdot \widehat{\sigma_{T_M}}$ where ${}^\bullet t_k \leq m_y$.*

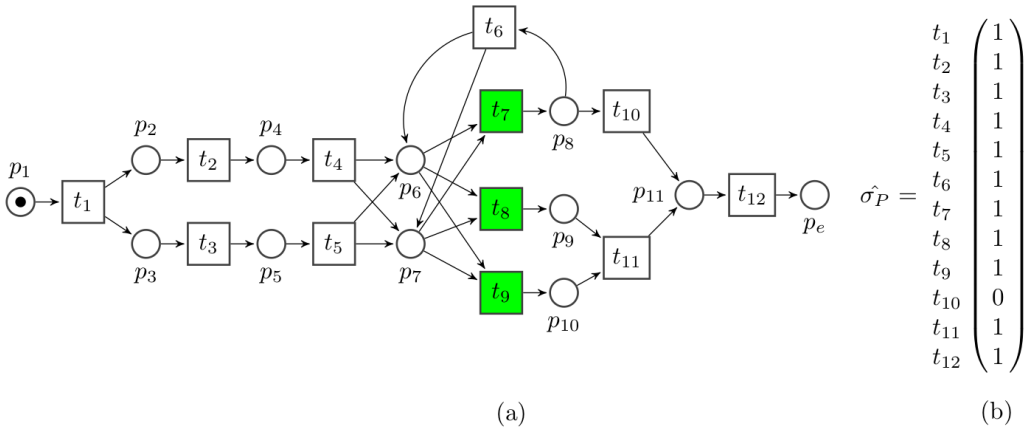


Fig. 9. (a) Process model N_5, (b) Parikh vector.

Lemma 4.5 states that given an arbitrary marking m_i, $\forall t_k \in T_{m_i} \cap supp(\widehat{\sigma_P})$, ${}^\bullet t_k$ will get or remains marked such that t_k be able to fire n_k times. To clarify this issue, a simple example is provided. Consider the model and $\widehat{\sigma_P}$ in Fig. 9 (a), (b). Take a closer look at $t_7, t_8, t_9 \in supp(\widehat{\sigma_P})$ with $\widehat{\sigma_P}[t_7] = \widehat{\sigma_P}[t_8] = \widehat{\sigma_P}[t_9] = 1$ which are highlighted. Based on Lemma 4.4 given that they are enabled, there exists at least one firing order which contains them, for example $t_8 t_7 t_9$ or $t_7 t_9 t_8$ are two firing orders. Note that according to Lemma 4.5 the set of places $\{p_6, p_7\} = {}^\bullet t_7 = {}^\bullet t_8 = {}^\bullet t_9$ remains or gets marked to let t_7, t_8 and t_9 get fired according to the mentioned firing sequences. At the end of this example it must be stressed that Lemma 4.4 and 4.5 state that for an arbitrary marking there is at least one order of firing to fire all transitions in $T_{m_i} \cap supp(\widehat{\sigma_P})$ according to their occurrences. The proof of Theorem 4.3 based on Lemmas 4.4, 4.5 is provided in Appendix A.2.

In Def. 4.2 if t_k was unable to be fired legitimately, another candidate in T_S which fulfills Def. 4.2, will be selected and fired. Therefore, t_k would be the last transition to be fired for that marking. As an example, consider again the model in Fig. 8, based on the provided marking, t_3 and t_{10} are enabled, $D(t_3, p_e) = 7$ and $D(t_{10}, p_e) = 5$, and ${}^\bullet t_3 = {}^\bullet t_{10}$, but because of $D(t_3, t_{10}) = \infty$, i.e., there is no direct path from t_3 to t_{10}, the former is unable to be fired legitimately, therefore t_{10} is fired instead.

The replaying approach presented in this paper avoids the necessity of backtracking, guarantees that the policy based on the legitimate firing will not miss the firing of any enabled transition encountered during the process. Nonetheless, for a given system net SN and $\widehat{\sigma_P}$, regardless of replaying approach the final marking m_{end} is reachable. More formally it is stated as follow:

[4]This corollary can be rephrased for $n_k > 1$ easily, but to keep things simple it was presented for $n_k = 1$.

Theorem 4.6 (Reachability of the final marking). *Let $\widehat{\sigma_P}$ be the Parikh vector which is computed based on Eq. (4) for a given system net $SN = (N, m_{start}, m_{end})$, then $\exists \sigma_R$ such that $\widehat{\sigma_R} \leqq \widehat{\sigma_P}$ and $m_{start}[\sigma_R\rangle m_{end}$.*

The proof of Theorem 4.6 in provided in Appendix A.3. It is worth stressing again that the existence of a firing sequence given a Parikh vector is only guaranteed for weakly sound bounded free-choice models, and not for more general Petri net classes. A representative example is as follows, consider the the model in Fig. 10 which is not a FC-net, and an observed trace $\sigma = a_1a_2a_8a_9a_6$, the computed Parikh vector based on Eq. 4, i.e., $\widehat{\sigma_P}$, contains t_1, t_2, t_8, t_9, t_6 which is not realizable in the net. One can see that despite of having a feasible solution for Eq. 4, i.e., $\widehat{\sigma_P}$, it is unable to be fired and reach us from initial marking p_1 to final marking p_6. Indeed whenever places p_3 and p_4 get marked then t_8 and t_9 can fire spuriously, namely they make negative marking for p_7 and p_8 and fill in these transitions at the same time.

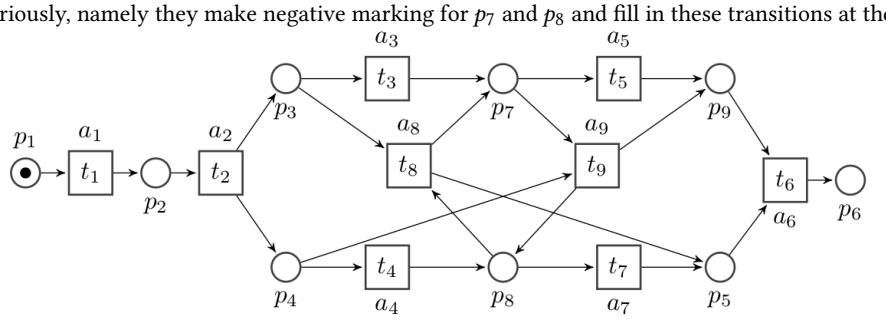


Fig. 10. Limitation for non free-choice models.

The mechanics of executing $\widehat{\sigma_P}$ are demonstrated by Alg. 2, 3 and 4. The global variables are: $\widehat{\sigma_P}$ and its support, which are computed as shown in the previous section, the distance matrix D and the system net, SN. Alg. 2, for a given marking, i.e., m_{curr}, identifies the farthest transition and its siblings, i.e., t_k and T_S, among the enabled set of transitions in $supp(\widehat{\sigma_P})$, line (5-7). If $T_S = \phi$ then t_k is fired simply and the current marking, m_{curr}, and $\widehat{\sigma_P}$ will be updated accordingly, line (8-10), otherwise it is examined whether it can be fired legitimately, line (12-15). If t_k was unable to be fired legitimately, other candidates in T_S are examined in the same way until one could be fired legitimately, line (17-23). Alg. 2 continues this procedure until it reaches the final marking, hence there is a loop in line 4. Notice that silent transitions, i.e., $\ell(t) = \tau$, are removed from the computed σ_N since they are invisible.

At the end one can see that for Alg. 2 the distance matrix D is computed once for a given model and event log. So the time complexity is $Max\{\Theta(|P| + |T|)^3, \Theta(\|\widehat{\sigma_P}\|_1)\}$ where the first term denotes the time complexity of computing D by Floyd - Warshall algorithm which relays on the number of places and transitions in the model, and the second term represents the number of elements in the Parikh vector $\widehat{\sigma_P}$ which are supposed to be replayed.

Algorithm 2 EXECUTE $\hat{\sigma_P}$

Input: Global Variables $supp(\hat{\sigma_P})$, D, $SN(N, m_{start}, m_{end})$
$\sigma_N \leftarrow \phi$ ▷ Initialize the modeled trace
$m_{curr} \leftarrow m_{start}$
while $(supp(\hat{\sigma_P}) \neq \phi \wedge m_{curr} \neq m_{end})$ **do**
  $T_C \leftarrow \{t \in T | {}^\bullet t \leq m_{curr}[p]\}$ ▷ Enabled transitions in the m_{curr}
  $T_{m_i} \leftarrow T_{m_i} \cap supp(\hat{\sigma_P})$ ▷ Enabled transitions in $supp(\hat{\sigma_P})$
  $T_S, t_k \leftarrow$ MAX_DIST(T_{m_i}) ▷ Finding the farthest transition and its siblings
  if $T_S = \phi$ **then**
    FIRE_UPDATE(t_k) ▷ Firing and updating the marking of the model
    $\sigma_N \leftarrow \sigma_N t_k$ ▷ Concatenating the fired transition
  else
    Paths$\leftarrow \{c \in \mathbb{N} | \exists t_j \in T_S, D(t_k, t_j) = c\}$
    if Paths $\neq \phi$ **then** ▷ Examine the legitimacy definition
      FIRE_UPDATE(t_k)
      $\sigma_N \leftarrow \sigma_N t_k$
    else ▷ Violating the legitimacy definition
      while Paths $= \phi$ **do** ▷ Looking for another transition, i.e., sibling of t_k to follow the legitimacy definition
        $T_S \leftarrow T_S \backslash \{t_k\}$
        $T_S, t_k \leftarrow$ MAX_DIST(T_S)
        Paths$\leftarrow \{c \in \mathbb{N} | \exists t_j \in T_S, D(t_k, t_j) = c\}$
      end while
      FIRE_UPDATE(t_k)
      $\sigma_N \leftarrow \sigma_N t_k$
    end if
  end if
end while
Return σ_N

Algorithm 3 MAX_DIST

Input: T_{in} ▷ T_{in}, is a set of transitions
$t_k \leftarrow \{t \in T_{in} | \forall t_i \in T_{in}, D(t) > D(t_i)\}$ ▷ Farthest transition, i.e, t_k
$T_S \leftarrow \{t_i \in T_{in} | {}^\bullet t_i = {}^\bullet t_k\}$ ▷ Siblings of a_{max}
Return T_S, t_k

Algorithm 4 FIRE_UPDATE

Input: t_k
Initialize X with 0, $X[t_k] \leftarrow 1$
$m_{next} = m_{curr} + \mathbf{N} \cdot X$
$\hat{\sigma_P}[t_k] \leftarrow \hat{\sigma_P}[t_k] - 1$ ▷ Updating $\hat{\sigma_P}$
Update $supp(\hat{\sigma_P})$
$m_{curr} \leftarrow m_{next}$ ▷ Updating the current marking
Return

4.2.1 The feasibility of executing $\widehat{\sigma_P}$. It is worth to point out that in the procedures just mentioned for executing $\widehat{\sigma_P}$ to obtain σ_N, $\widehat{\sigma_N} \leqq \widehat{\sigma_P}$. This is due to the fact that computing $\widehat{\sigma_P}$ by the formulation in Eq. (4) relays on marking equation, i.e., Eq. (1), therefore on occasions it may have spurious elements, i.e., those which are not reachable during replaying. Therefore in the worst case no complete modeled trace σ_N with the following condition i.e., $\widehat{\sigma_N} = \widehat{\sigma_P}$, is guaranteed to be generated neither by the proposed approach nor other replaying techniques. However based on the following Theorem, the proposed approach always finds a sequence like σ_N such that $m_{start}[\sigma_N\rangle m_{end}$.

Theorem 4.7 (Existence of the modeled trace). *For the proposed replaying approach given the contexts of Def. 4.1, 4.2, $\exists\sigma_N$ where $\widehat{\sigma_N} \leqq \widehat{\sigma_P}$ such that $m_{start}[\sigma_N\rangle m_{end}$.*

The proof of Theorem 4.7 is presented in Appendix A.4. Theorem 4.7 states that the proposed technique finds a sequence like σ_N by which m_{end} is reachable from m_{start}, in other words the technique is complete. The following theorem proves that σ_N is the longest sequence among the existing sequences by which the final marking is reachable from the initial marking.

Theorem 4.8 (Length optimality of modeled trace). *Given the context of Theorem 4.7, $\nexists\sigma'_N$ where $\widehat{\sigma_N} \leqq \widehat{\sigma'_N} \leqq \widehat{\sigma_P}$ and $|\sigma_N| < |\sigma'_N|$ such that $m_{start}[\sigma'_N\rangle m_{end}$.*

The proof of Theorem 4.8 is given in Appendix A.5. According to the above theorem since $\widehat{\sigma_N} \leqq \widehat{\sigma_P}$ then some elements of $\widehat{\sigma_P}$ are spurious, i.e., never get enabled, and hence will not be fired. This is regardless of having well-formed or not WF-net models, but for unstructured models, i.e., *Spaghetti*, $\widehat{\sigma_P}$ may have in general more spurious elements. To enlighten this issue consider the model N_6 in Fig. 11 (a) and observed trace $\sigma = t_1t_2t_5t_7t_6t_8t_{10}$. The computed Parikh vector $\widehat{\sigma_P}$ is represented in Fig. 11 (b). One can see that $\widehat{\sigma_P}[t_5] = \widehat{\sigma_P}[t_6] = \widehat{\sigma_P}[t_7] = 1$ whereas the corresponding counterparts in $\widehat{\sigma_N}$ are zero. Those spurious elements (see Sect. 3.2) never get enabled while $\widehat{\sigma_P}$ is being replayed; also note that they do not violate constraints of Eq. (4) while $\widehat{\sigma_P}$ is computed.

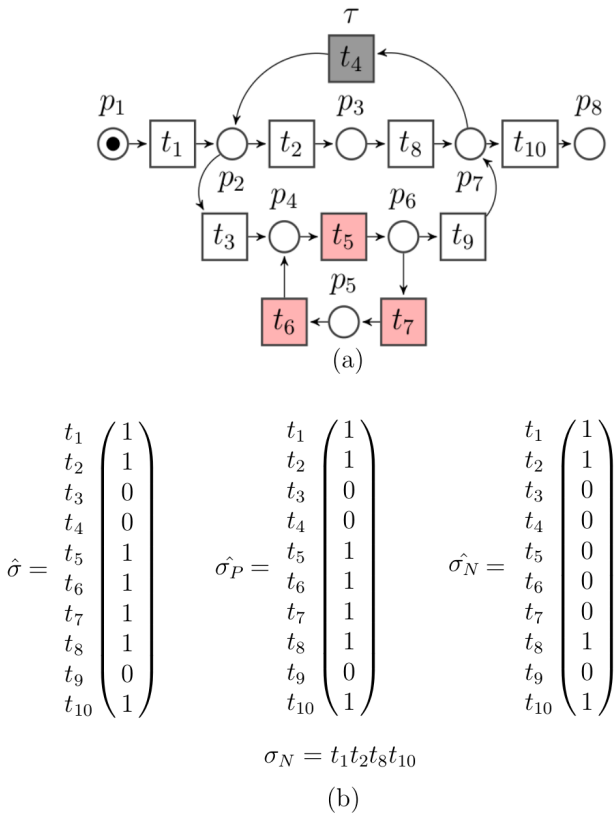


Fig. 11. (a) Process model N_6, (b) Related Parikh vectors.

4.3 Aligning σ and σ_N

This subsection is centered around aligning a model trace σ_N, computed with the strategy described in the previous section, and an observed trace σ. The computed alignment is called initial or base alignment, α_b. It is called initial, since its fitness value might be improved by rearranging some parts. Informally, we will proceed in two steps: first, we will treat σ and σ_N as mere words, and will align them through a *sequence alignment* technique. Since the obtained sequence alignment allows for illegal moves (c.f. Def. 3.6), it is converted into an alignment by transforming each illegal move into a pair of asynchronous moves. Let us first start by defining the alignment of sequences:

Definition 4.9 (Alignment of Sequences). Assume that S represents the alphabet and let S_A and S_B be two members of S^+ with length m and n respectively. Let $\perp$ denote the empty set, then:

- (A, B) is a *match*, if $A \in S_A$, $B \in S_B$ and $A = B$
- (A, B) is a *mismatch*, if $A \in S_A$, $B \in S_B$ and $A \neq B$
- (A, B) is a *gap*, if $A = \perp$ and $B \in S_B$
- (A, B) is a *gap*, if $A \in S_A$ and $B = \perp$

If S_M contains all possible pairs of elements as defined above, then given a sequence alignment $\alpha \in S_M^*$, the projection of the first element (ignoring $\perp$) results in S_A, and projecting the second element (ignoring $\perp$) results in S_B.

In Def. 4.9, match and gap are the same as synchronous and asynchronous moves in Def. 3.6, but mismatch has no counterpart in the latter. So after aligning modeled and observed trace, i.e., σ_N and σ, by this method, if a mismatch pair (X, Y) with $X \in \sigma, Y \in \sigma_N$ and $\ell(Y) \neq X$ occurs, then it will be transformed to $(X, \perp)$ and $(\perp, Y)$ to preserve properties of Def.3.6.

We use a well-known technique based on dynamic programming for obtaining a sequence alignment between σ_N and σ [Needleman and Wunsch 1970]. This technique builds up the solution by determining all similarities between arbitrary prefixes of the two sequences. The algorithm starts with shorter prefixes, and uses previously computed results to solve the problem for larger prefixes. It attempts to maximize similarity between the two input sequences by employing a scoring matrix to penalize gaps and mismatches among them. The scoring matrix can be obtained by aligning the corresponding two sequences. Formally, we use the notion of alignment between two *sequences* presented in [Neapolitan 2014], which is based on the seminal work in [Needleman and Wunsch 1970].

To this end, a primary matrix is created, where the first row and column are filled by observed and modeled traces respectively, as depicted in Fig. 12 (a). The second row and the second column are initialized with numbers starting from 0,-1,-2,..., they are depicted in yellow color in the figure. The task then is to fill the remaining cells with the recurrence Eq. (5), in which δ represents the gap penalty, and $s(t_i, a_j)$ represents both the match and mismatch cost between two elements t_i and a_j which are modeled and observed trace elements, respectively[5].

$$SIM(t_i, a_j) = MAX \begin{cases} SIM(t_{i-1}, a_{j-1}) + s(t_i, a_j) \\ SIM(t_{i-1}, a_j) - \delta \\ SIM(t_i, a_{j-1}) - \delta \end{cases} \quad \text{, where} \quad s(t_i, a_j) = \begin{cases} \beta & \text{If} \quad \ell(t_i) = a_j \\ -\beta & \text{If} \quad \ell(t_i) \neq a_j \end{cases} \tag{5}$$

To enlighten of how to fill the scoring matrix, consider $\sigma_N = t_1 t_4 t_5 t_6$ with $\ell(t_1) = a_1, \ell(t_4) = a_4, \ell(t_5) = a_5, \ell(t_6) = a_6$ and observed trace $\sigma = a_1 a_4 a_6$. Also assume $\delta = 1$ and $\beta = 1$. Generally speaking, by the approach of [Needleman and Wunsch 1970] filling the primary matrix is as follows: start from the up left corner as depicted in Fig. 12 (b). Move through the cells row by row, calculating the score for each cell by Eq. (5). The score is calculated as the best possible score (i.e. highest) from existing scores to the left, top or top-left (diagonal). When a score is calculated from the top, or from the left this represents a gap, see Fig. 12 (c). When the score is calculated from the diagonal, this represents either a math or a mismatch. In case of a mismatch, this will be treated in the next stage. The final scoring matrix is depicted in Fig. 12 (d). Once it is computed, given δ and β, the bottom right entry of the matrix gives the maximum score, i.e., the number which indicates maximum similarity among the given strings, among all possible sequence alignments.

[5]At this point, it must be stressed that the costs used in the algorithm for sequence alignment are not related to the cost for an alignment, as defined in Def. 3.7.

(a)

		a_1	a_4	a_6
	0	-1	-2	-3
t_1	-1			
t_4	-2			
t_5	-3			
t_6	-4			

(b)

		a_1	a_4	a_6
	0	-1	-2	-3
t_1	-1	1		
t_4	-2			
t_5	-3			
t_6	-4			

$$s(t_1, a_1) = 1$$
$$SIM(t_1, a_1) = MAX \begin{cases} SIM(0,0) + s(t_1, a_1) = 1 \\ SIM(t_1, 0) - 1 = -2 \\ SIM(0, a_1) - 1 = -2 \end{cases}$$

(c)

		a_1	a_4	a_6
	0	-1	-2	-3
t_1	-1	1	0	
t_4	-2			
t_5	-3			
t_6	-4			

$$s(t_1, a_4) = -1$$
$$SIM(t_1, a_4) = MAX \begin{cases} SIM(0, a_1) + s(t_1, a_4) = -2 \\ SIM(t_1, a_1) - 1 = 0 \\ SIM(0, a_4) - 1 = -3 \end{cases}$$

(d)

		a_1	a_4	a_6
	0	-1	-2	-3
t_1	-1	1	0	-1
t_4	-2	0	2	1
t_5	-3	-1	1	0
t_6	-4	-2	0	2

$$s(t_6, a_6) = 1$$
$$SIM(t_6, a_6) = MAX \begin{cases} SIM(t_5, a_4) + s(t_6, a_6) = 2 \\ SIM(t_6, a_4) - 1 = -1 \\ SIM(t_5, a_6) - 1 = -1 \end{cases}$$

Fig. 12. (a) Primary matrix, (b) and (c) Filling the matrix, (d) Final scoring matrix

		a_1	a_4	a_6
	0	-1	-2	-3
t_1	-1	1	0	-1
t_4	-2	0	2	1
t_5	-3	-1	1	0
t_6	-4	-2	0	2

(a)

$\alpha_b =$

a_1	a_4	$\perp$	a_6
t_1	t_4	t_5	t_6

(b)

Fig. 13. (a) Trace back of the scoring matrix, (b) The computed alignment

To compute the alignment that corresponds to this sequence alignment, we start from the bottom right entry, and compare the value with three possible sources, i.e., top, left and diagonal to identify from which one of them it came from. If it was fed by a diagonal entry, either it represents a synchronous move between corresponding elements, or a mismatch. In the later case, it will be decomposed to $(A, \perp)$ and $(\perp, B)$ to satisfy Def. 3.6. If it was fed by a top or left entries, then it represents an asynchronous move, see Fig. 13 (a). Following the above described steps, the alignment of two sequences can be found which represented in Fig. 13 (b). Note that given δ and β, the score of α_b is (+1)+(+1)+(-1)+(+1)=2. Also based on Def. 3.8 by assuming $\delta_S = 1, \delta_L = \delta_M = 2$, α_b has the fitness value, 1-2/5 or 60%. Notice that in some situations there may be two or more possible alignments between the two sequences which has the same maximum score, i.e., α_b is not unique.

The corresponding procedures for computing α_b is represented in Alg. 5 and 6. Alg. 5 starts to compute the scoring matrix M for given σ_N and σ by initializing it, lines 4-9. Then, it fills each entry according to Eq. (5), lines 11-21, and at the same time for each entry books the source of the computed score, i.e., *top, left or diagonal*, in matrix S, line 21. Alg. 6 to obtain α_b uses the source matrix S to trace back by starting at its right bottom element $S[|\sigma_N|, |\sigma|]$, lines 3-4, to find a path to the top left element $S[1, 1]$, lines 5-16. According to the content of the element under consideration, the corresponding synchronous or asynchronous moves are added to α_b. This procedure continues until it reaches the top left element. The time complexity of computing α_b is related to the computation of scoring matrix M, which is $\Theta(|\sigma_N| * |\sigma|)$.

Algorithm 5 Align σ, σ_N

```
1:  Input: σ, σ_N, ℓ, β, δ                ▷ ℓ is the labeling functions of transitions in σ_N
2:  M ← φ                                 ▷ M is the primary scoring matrix with dimension (|σ_N| + 1) * (|σ| + 1)
3:  S ← φ                                 ▷ S is the source matrix with dimension (|σ_N|) * (|σ|) which books the source of scores
4:  for i ← 0 to |σ| do                   ▷ Initializing the first row of M
5:      M[0, i] ← −i
6:  end for
7:  for j ← 0 to |σ_N| do                 ▷ Initializing the first column of M
8:      M[j, 0] ← −j
9:  end for
10: ————————————————————
11: for j ← 1 to |σ_N| do                 ▷ Filling the scoring matrix M
12:     for i ← 1 to |σ| do
13:         if (ℓ(σ_N[j]) = σ[i]) then    ▷ Match between elements
14:             s(σ_N[j], σ[i]) ← β
15:         else                          ▷ Mismatch between elements
16:             s(σ_N[j], σ[i]) ← −β
17:         end if
18:         diag ← M[j − 1, i − 1] + s(σ_N[j], σ[i])
19:         top ← M[j − 1, i] − δ
20:         left ← M[j, i − 1] − δ
21:         M[j, i], S[j, i] =Max(diag, top, left)   ▷ Computing the maximum score and the name of the source, i.e., diag, left, top
22:     end for
23: end for
24: Return M, S
```

Algorithm 6 Traceback

Input: S ▷ S is the source matrix computed by previous algorithm
$\alpha_b \leftarrow \phi$
$j \leftarrow |\sigma_N|$ ▷ Trace back from the bottom right element of S
$i \leftarrow |\sigma|$
while $(i \neq 0 \vee j \neq 0)$ **do**
  if $(S[j,i]$ =diag) **then** $\alpha_b \leftarrow \alpha_b(\sigma[i], \sigma_N[j])$ ▷ Synchronous move
    $i \leftarrow i - 1$
    $j \leftarrow j - 1$
  end if
  if $(S[j,i]$ =top) **then** $\alpha_b \leftarrow \alpha_b(\perp, \sigma_N[j])$ ▷ Move in model
    $j \leftarrow j - 1$
  end if
  if $(S[j,i]$ =left) **then** $\alpha_b \leftarrow \alpha_b(\sigma[i], \perp)$ ▷ Move in log
    $i \leftarrow i - 1$
  end if
end while
————————————————————
for (u, v) in α_b **do** ▷ Standardizing α_b according to Def. 3.6
  if (u, v) is a mismatch **then**
    Replace (u, v) with $(u, \perp), (\perp, v)$
  end if
end for
Return α_b

4.4 Fitness improvement by local search

Although sequence alignment is guaranteed to be optimal, the alignment α_b arising from the sequence alignment is not guaranteed to be the best one by which the model mimics the observed trace. The root cause of this problem comes from the fact that there may be several traces which can be obtained by executing the computed Parikh vector $\widehat{\sigma_P}$. Therefore the initial computed alignment α_b in sub-Sec. 4.3, might be far away from what is desired, i.e., the optimal one.

This section is centered around *reordering* the initial alignment α_b, with the aim of improving the corresponding fitness value. This is obtained by trying to increase the number of synchronous moves between observed and modeled traces, given that the modeled trace σ_N remains executable. More rigorously, the reordering of α_b can be accomplished by a local search approach in which a move in log, i.e. $(a_i, \perp)$, will be merged by the corresponding move in model, i.e., $(\perp, t_j)$ and $\ell(t_j) = a_i$, to make a synchronous move, i.e., (a_i, t_j). This can only be done if: first, the resulted modeled trace remains executable, and second, the order of events in the observed trace remains unchanged. The iterative approach stops once no more merging can be made. To clear up this idea, first the formal definitions will be presented and then an illustrative example will be provided.

Definition 4.10 (Alignment reordering). Given a system net $SN(N, m_{start}, m_{end})$ and an alignment α. Let $\sigma_\perp$ and $\sigma_{N,\perp}$ be the projection of α onto the first and second elements (including $\perp$), respectively. The move in log $(a_i, \perp)$ with $a_i = \sigma_\perp[i]$, can be merged with the move in model $(\perp, t_j)$ with $t_j = \sigma_{N,\perp}[j]$, to make the synchronous move (a_i, t_j) given that the following conditions are met:

- $\ell(t_j) = a_i$

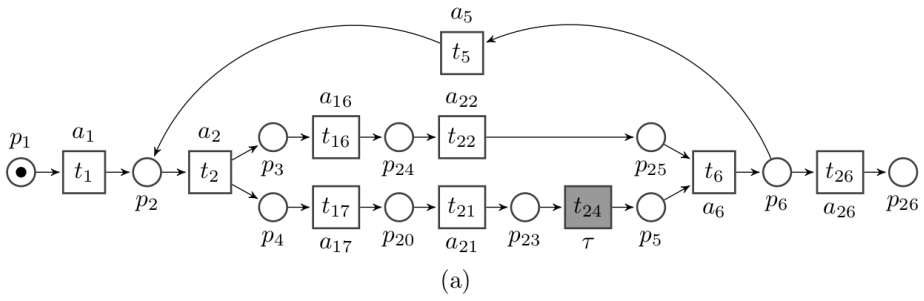
- After merging, the new modeled trace $\sigma'_{N,\perp}$ with the following changes

$$\sigma'_{N,\perp}[k] = \begin{cases} \sigma_{N,\perp}[k] & \text{if } k \neq i, j \\ \perp & \text{if } k = j \\ t_j & \text{if } k = i \end{cases}$$

must be executable, i.e., $(N, m_{start})[\sigma'_{N,\perp}\rangle(N, m_{end})$.

Merging two moves that satisfy Def. 4.10 will contribute to a fitness increase, since the number of synchronous moves will rise and the number of asynchronous moves will decrease. As implicitly stated in Def. 4.10, it must be emphasized that, after merging, the order of events in the given observed trace, i.e., σ, will not be changed.

Based on Def. 4.10, the reordering of the initial alignment α_b can be done in an iterative way as long as no more synchronous moves can be obtained, hence the arc back from fitness improvement part to computing alignment part in Fig. 2.



(a)

$\alpha_b =$

a_1	a_2	a_{16}	$\perp$	a_{22}	$\perp$	a_{21}	a_{17}	$\perp$	$\perp$	a_6	a_{16}
t_1	t_2	$\perp$	t_{17}	$\perp$	t_{16}	t_{21}	$\perp$	t_{22}	t_{24}	t_6	t_{26}

(b)

$\alpha_b =$

a_1	a_2	a_{16}	$\perp$	a_{22}	a_{21}	a_{17}	$\perp$	a_6	a_{16}
t_1	t_2	t_{16}	t_{17}	t_{22}	t_{21}	$\perp$	t_{24}	t_6	t_{26}

(c)

Fig. 14. (a) Process model N_7, (b) Initial alignment α_b, (c) Alignment after reordering

To illustrate how the reordering of an alignment works, a simple example is presented. Consider the model and initial alignment α_b in Fig. 14 (a), (b) respectively. The appropriate asynchronous moves which are candidate for merging and deriving new synchronous moves are highlighted with the same color in Fig. 14 (b)[6]. One can see that $(a_{16}, \perp)$ and $(\perp, t_{16})$, highlighted in green, can be merged to make the synchronous move (a_{16}, t_{16}), and the resulted modeled trace remains executable. The same holds for moves $(\perp, t_{22})$, $(a_{22}, \perp)$, highlighted in blue, to make (a_{22}, t_{22}). In contrast,

[6]Please note that t_{24} is an invisible transition hence not considered as a candidate, and therefore it is not highlighted.

the two asynchronous moves $(\perp, t_{17})$, $(a_{17}, \perp)$ can not be merged since otherwise the resulted modeled trace is not executable.

4.4.1 A large example. This subsection presents a large example, to shed light on how the fitness value of and initial alignment, α_b, can be significantly improved by the reordering technique in Def. 4.10 just mentioned in the previous section.

Consider the model in Fig. 15 (a), observed and modeled traces σ, σ_N in Fig. 15 (b). σ_N was computed by the method in Sec. 4.2. Notice that in the model, for the sake of simplicity, the subscript of each transition and its label are the same except those highlighted, which represent transitions with the same label. The initial alignment α_b, its fitness π_{α_b}, and $\sigma_{N,\perp}$ is depicted in Fig. 16 (a), (b). For the moment we ask the reader to ignore highlighted parts.

The initial α_b is far from the optimal alignment. As mentioned already, this problem might be somewhat alleviated by reordering α_b, i.e., by merging the appropriate moves in log and model, to improve the corresponding fitness and preserve the executable property of the modeled trace simultaneously.

For example in Fig. 16 (a), the asynchronous moves $(a_3, \perp)$ and $(\perp, t_3)$, highlighted in green color, can be merged to make a synchronous move (a_3, t_3), highlighted with the same color in Fig. 16 (b). Notice that the resulted modeled trace remains executable. The same holds for asynchronous moves $(a_6, \perp), (a_8, \perp)$, which can be transformed to synchronous moves $(a_6, t_6), (a_8, t_8)$ by the corresponding merging which is highlighted accordingly. It is worth to note that $(a_{10}, \perp)$ and $(\perp, t_{10})$, highlighted with red, cannot be merged to (a_{10}, t_{10}), because in that case the resulting modeled trace is not executable. In contrast, $(a_{19}, \perp)$ and $(\perp, t_{18})$ can be merged without violating any conditions mentioned in Def. 4.10, as depicted in Fig. 16 (c) highlighted with gray. The same holds for $(a_{23}, \perp)$, which can be merged with its counterpart move in mode, i.e., $(\perp, t_{22})$. The corresponding merging is depicted in Fig. 16 (e). The last two merged moves reveal that transitions with the same label do not pose any challenges during the reordering of α_b. The final alignment and corresponding modeled trace, where no more reordering can be applied, are shown in Fig. 17 (a), (b).

To measure how much the fitness value is improved for the mentioned example, one can see that for the initial alignment α_b in Fig. 16 (a), based on Def. 3.8 with $\delta_S = 1, \delta_L = 2$, and $\delta_M = 2$, $\pi_{\alpha_b} = 1 - \frac{16\times2+18\times2}{11\times1+16\times2+18\times2} = 0.139$ and for the alignments depicted in Fig. 16 (c), (e) after merging highlighted moves are 0.188 and 0.25 respectively. Finally, the alignment α, see Fig. 17 (a), for which no more fitness improvement can be obtained has $\pi_\alpha = 1 - \frac{2\times2+4\times2}{25\times1+2\times2+4\times2} = 0.675$, which represents rather magnificent fitness improvement, even though it is not the optimal one. The optimal alignment α_{opt}, with $\pi_{\alpha_{opt}} = 0.764$, for the mentioned example provided by state of the art approach [Adriansyah 2014] is depicted in Fig. 17 (c). One can see that the only difference for this example between our approach and the approach in [Adriansyah 2014] is the synchronous move (a_{10}, t_{10}) in α_{opt}.

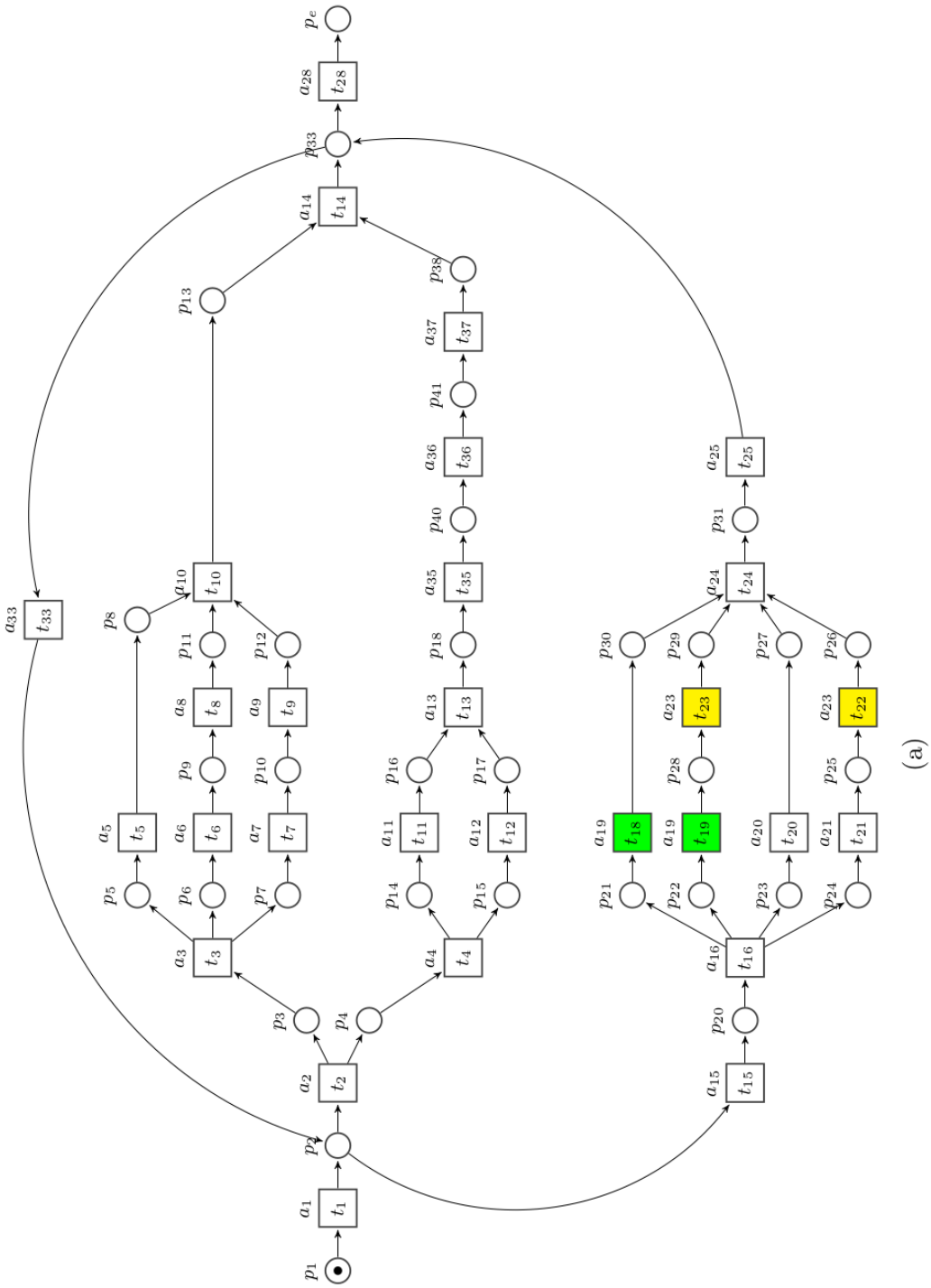


Fig. 15. (a) Process model N_8, (b) Observed trace σ and modeled trace σ_N with and without labels

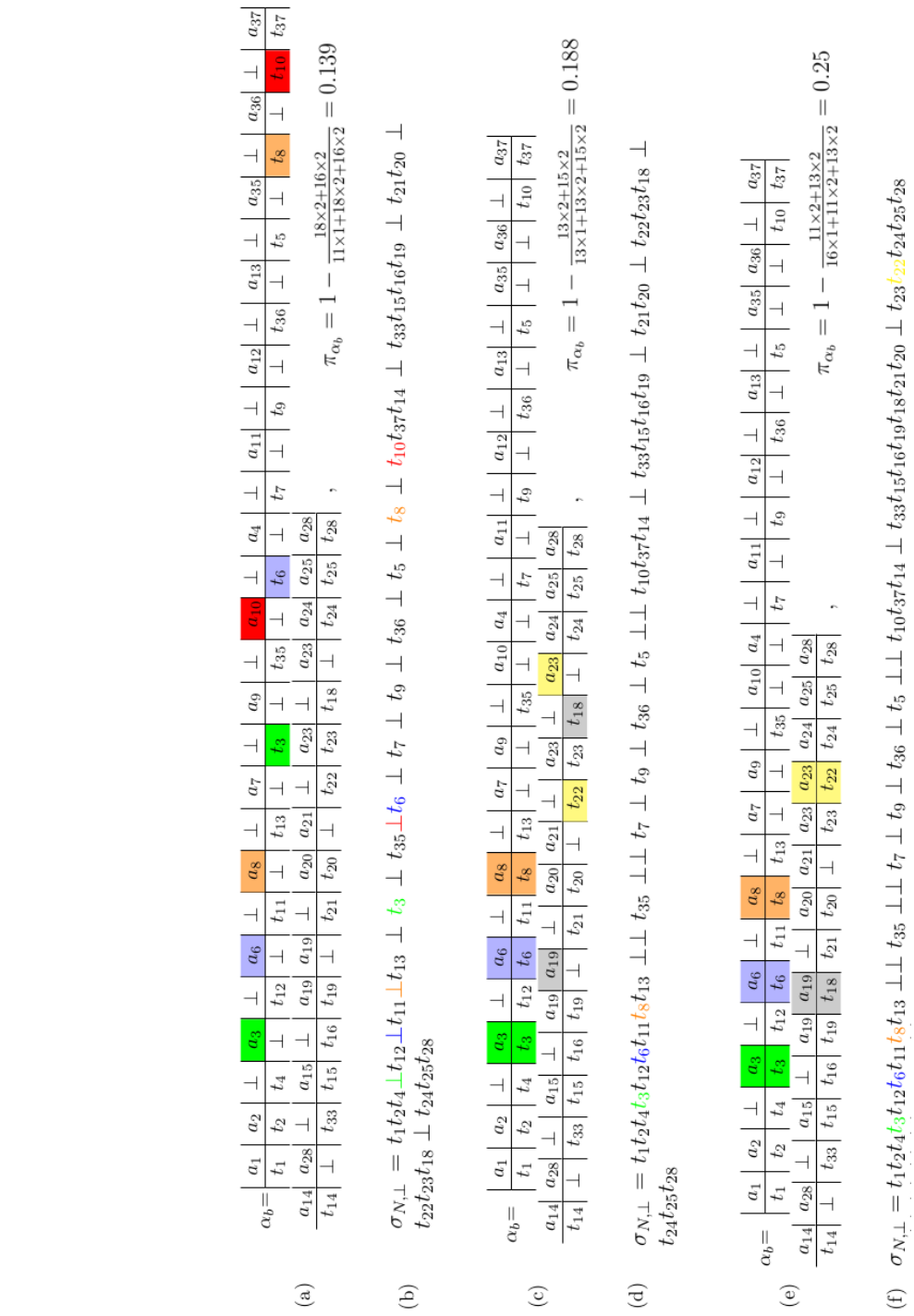


Fig. 16. (a) Initial alignment, (b) Initial modeled trace including ⊥, (c)-(f) Alignments and modeled trace after some adjustments

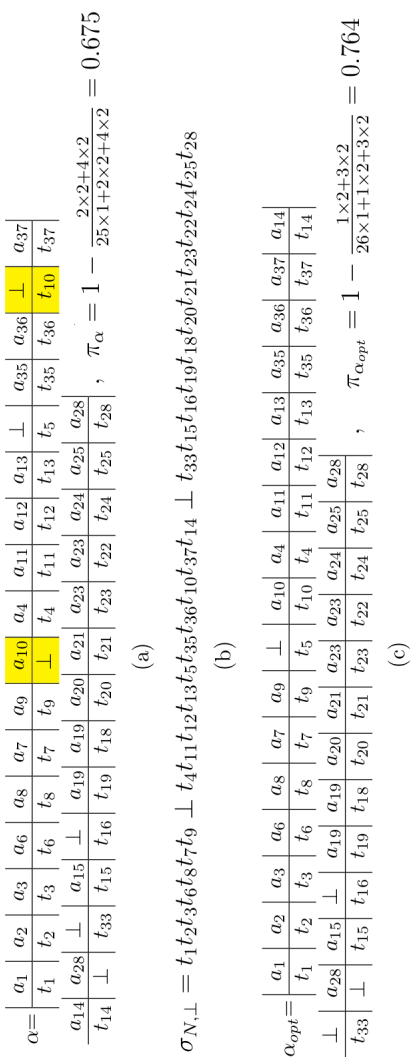


Fig. 17. (a) Final alignment, (b) Final modeled trace including $\perp$, (c) Optimal alignment provided by state of the art approach [Adriansyah 2014]

The corresponding procedures required for the reordering of an alignment α are depicted by Alg. 7 and 8. The input alignment will be considered as the initial alignment, α_b. Given α_b, for each move in log, $(a_i, \perp)$, Alg. 7 finds the corresponding move in model $(\perp, t_j)$ with $a_i = \ell(t_j)$ to be merged, then examines whether the synchronous move (a_i, t_j) can be made with the use of flipping corresponding elements, lines 5-8. Whenever the candidate synchronous move is created without violating the conditions in Def. 4.10, then it breaks the current loop, line 9, and starts the same

Algorithm 7 Alignment Reordering

```
1:  Input: Global Variables SN(N, m_start, m_end), α_b
2:  prev-fitness ← 0
3:  curr-fitness← π_{α_b}
4:  while (curr-fitness > prev-fitness) do                ▷ Iterate until no fitness improvement can be made
5:      for (a_i, ⊥) in α_b do                            ▷ Iterate over moves in log
6:          for (⊥, t_j) in α_b do                        ▷ Iterate over moves in model
7:              if (a_i = ℓ(t_j)) then                    ▷ Examine whether move on log and model match
8:                  if (Merge((a_i, ⊥),(⊥, t_j))) then    ▷ Merging corresponding asynchronous moves
9:                      Break                             ▷ Break the current loop and start for another move on log
10:                 end if
11:             end if
12:         end for
13:     end for
14:     prev-fitness ← curr-fitness
15:     curr-fitness ← π_{α_b}
16: end while
17: Return α_b
```

Algorithm 8 Merge

```
1:  Input: (a_i, ⊥),(⊥, t_j)                              ▷ Inputs are move in log and model respectively
2:  σ_{N,⊥}[i] ← t_j                                      ▷ Flipping the corresponding elements to make a synchronous move
3:  σ_{N,⊥}[j] ←⊥
4:  if (m_start[σ_{N,⊥}⟩m_end) then                       ▷ Examine whether the resulted modeled trace is executable
5:      Update π_{α_b}                                    ▷ Updating the fitness value based on the reordered α_b
6:      Return 1
7:  else                                                  ▷ Undo the flipping if σ_{N,⊥} is not executable
8:      σ_{N,⊥}[i] ←⊥
9:      σ_{N,⊥}[j] ← t_j
10:     Return 0
11: end if
```

procedure for the another move in log[7]. Merging corresponding asynchronous moves can be accomplished successfully if the resulted modeled trace σ_N is executable, Alg 8, line 4.

5 EXPERIMENTS

The proposed approach in this paper has been implemented in Python 2.7 as the prototype tool ILPSDP[8] and Gurobi [Gurobi Optimization 2016] was used as the LP solver. The standalone files for both Linux and Microsoft Windows operating systems can be downloaded from [Taymouri 2017]. The tool has been evaluated over different family of examples, from artificial to realistic, containing transitions with duplicate labels, and from well-structured to completely spaghetti. Well-structured models are similar in structure to the one shown in Figure 15(a), where one can see branching (e.g., choice at place p_2 between t_2 and t_{15}), concurrency (e.g., transitions t_5 and t_6 are concurrent), or loops (backward arc through transition t_{33}).

[7]It is worth to mention that the approach considered in Alg. 7 is the simplest one to clear-up the idea in an algorithmic way, obviously for extremely large observed traces incorporating a hash function to map corresponding moves would be more efficient.
[8]The experiments have been done on a desktop computer with Intel Core i7-2.20GHz, and 8GB of RAM.

The results of ILPSDP are compared with the state of the art techniques for computing alignments [Adriansyah 2014] (denoted A^*), the recent technique reported in [van Dongen 2018] (denoted Inc)[9], and the automata-based technique [Reißner et al. 2017] (denoted DAFSA). The specification of benchmark datasets selected from [Taymouri and Carmona 2016a], [Munoz-Gama et al. 2014], [Taymouri and Carmona 2016b] and [4dt 2018], are presented in Tables 1-3. In short each table contains the following information about each dataset:

- *Model Characteristics*:
  - $|P|$, $|T|$, and $|Arc|$: They represent the number of places, transitions, and arcs in the process model respectively.
  - *Ex.Cycl.*: This parameter, i.e., Extended Cyclomatic complexity, shows the count of potential paths through the model. The higher the count, the more complex the model [Lassen and van der Aalst 2009].
  - *Struct.*: The Structuredness metric, quantifies a WF-net in terms of basic structures like sequence, choice, iteration and etc, or in general the atomic patterns that are understood by the people [Lassen and van der Aalst 2009].
- *Event log Characteristics*:
  - *Cases(dist.)*: This column shows the number of cases in an input event log. Since a case might happen several times, the number of distinct cases are mentioned too.
  - *Fitting*: This column indicates whether the corresponding cases are executable.
  - $|\sigma|_{avg}$: This column indicates the average length of observed traces (cases).
  - $|Events|$: This column represents the number of events in an input event log.

In more detail, all the models in Table 1 are acyclic and without invisible transitions. Event logs of prAm6, prBm6, prEm6, prFm6 and prDm6 are very fitting. In contrast, prCm6 contains traces with many deviations. Models in Table 2, are well-structured, cyclic and have invisible transitions. Models in Table 3 are cyclic and with invisible transitions, *Documentflow1* and *Documentflow2* are spaghetti models (see the corresponding structuredness values), and do have small numbers of understandable atomic parts. The remaining models are well-structured.

In addition„ to asses how the proposed approach can deal with instances containing duplicate transitions, a set of benchmarks of this type, classified by different amount of errors, 25%, 35%, 50%, 75%, were generated by PLG2 [Burattin 2016][10] These numbers denote the extent of deviations (making a fitting trace into a non-fitting trace. This can be done by inserting or removing events, and reshuffling the current events) in each event log. The details of these datasets are reported in Table 4.

[9]The approach [van Dongen 2018] was not accessible at the time of writing this paper, so we include for the sake of reference the numbers for these benchmarks that are reported in the aforementioned paper. In addition„ according to that paper, the experiments were conducted in single-threaded mode on a 2.8 GHz Intel Xeon W3530 CPU.

[10]At the time of generating models for the experiments, PLG2 in fact was unable to produce models containing duplicate labels from scratch, therefore the generated models and logs were modified in order to have duplicate transitions.

Table 1. BPM2013 artificial benchmark datasets [Munoz-Gama et al. 2014],[Taymouri and Carmona 2016b]

Process model						Event log			
Name	$\|P\|$	$\|T\|$	$\|Arc\|$	Ex.Cycl.	Struct.	Cases (dist.)	Fitting	$\|\sigma\|_{avg}$	$\|Events\|$
prAm6	363	347	846	10001	3156	1200 (1049)	No	31	37961
prBm6	317	317	752	3098	2354	1200 (1126)	Yes	43	49792
prCm6	317	317	752	3098	2354	500 (500)	No	43	21465
prDm6	529	429	1140	10003	2840	1200 (1200)	No	248	298322
prEm6	277	275	652	10003	2532	1200 (1200)	No	98	118513
prFm6	362	299	772	10002	2843	1200 (1200)	No	240	288931
prGm6	357	335	826	10003	3557	1200 (1200)	No	143	171685

Table 2. Artificial benchmark datasets [Taymouri and Carmona 2016b], [Taymouri and Carmona 2016a]

Process model						Event log			
Name	$\|P\|$	$\|T\|$	$\|Arc\|$	Ex.Cycl.	Struct.	Cases (dist.)	Fitting	$\|\sigma\|_{avg}$	$\|Events\|$
M_1	40	39	92	341	1284	500 (453)	No	13	6555
M_2	34	34	80	672	1351	500 (500)	No	17	8809
M_3	108	123	276	900	6355	500 (462)	No	37	17980
M_4	36	52	106	21	127	500 (496)	No	26	13421
M_5	35	33	78	9321	314	500 (500)	No	34	17028
M_6	69	72	168	9857	2225	500 (500)	No	53	26719
M_7	65	62	148	9948	10479	500 (500)	No	37	18803
M_8	17	15	36	10	66	500 (432)	No	17	8246
M_9	47	55	120	60	271	500 (500)	No	44	22163
M_{10}	150	146	354	10011	6234	500 (500)	No	58	29118

Table 3. Realistic benchmark datasets [Taymouri and Carmona 2016b], [Taymouri and Carmona 2016a], [4dt 2018]

Process model						Event log			
Name	$\|P\|$	$\|T\|$	$\|Arc\|$	Ex.Cycl.	Struct.	Cases (dist.)	Fitting	$\|\sigma\|_{avg}$	$\|Events\|$
Bank.	121	114	272	289	2371	2000 (2000)	No	58	116839
Doc1.	334	447	2059	9906	$6.3E8$	12391 (1411)	No	5	65653
Doc2.	337	456	2025	9919	$7.6E8$	12391 (1411)	No	5	65653
Traffic.	15	23	48	22	156	10000 (231)	No	4	561470
BPI2017	134	279	558	152	11320	31509 (15930)	No	38	1202226
BPI2018	228	550	1100	330	20142	43809 (28457)	No	57	2514266

Table 4. Models with duplicate transitions

Process model						Event log				
Name	$\|P\|$	$\|T\|$	$\|Arc\|$	Ex.Cycl.	Struct.	Cases (dist.)	Fitting	$\|\sigma\|_{avg}$	Duplicate Trans.	$\|Events\|$
ML_1	27	35	74	24	106	500 (499)	No	28	2	14474
ML_2	165	177	404	9299	533027	500 (500)	No	87	12	43890
ML_3	45	45	106	94	2550	500 (499)	No	26	2	13251
ML_4	36	33	80	3646	2330	500 (500)	No	28	6	14074
ML_5	159	172	390	3405	27317	500 (284)	No	42	14	21026

Execution Times Comparison. Fig. 18 (a) and Fig. 18 (b), (c), (d) provide the required execution times for the mentioned datasets by ILPSDP, A^*, Inc and DAFSA. As stated before, for Inc only a subset of the benchmarks was reported, due to the tool not being currently available.

One can see that for all benchmark datasets except M_8, M_9, ML_1 and *Banktransfer (Bank.)*, ILPSDP is faster than A^*. For the benchmarks where A^* is better, the effort to solve an ILP model and then apply a local search is greater than simply exploring the state-space of the model, since these instances are either small or contain very few deviations. Notice that Inc is also very competitive for the models that are well-structured. For DAFSA, when the state space becomes large (e.g., *Documentflow* or most of the rest of the examples), it cannot handle them. Hence, competency of ILPSDP becomes more clear in dealing with big models and long traces where the other approaches need to explore a large search space. As an example, for *prDm6* A^* runs out of memory (N/A in the figure), and ILPSDP can accomplish the task in reasonable time. By the same token, for realistic datasets ILPSDP is faster on *spaghetti* or *unstructured* models, even when there are many loops, see the results for *Documentflow1, Documentflow2*.

Also, the required execution time for ILPSDP is not as sensitive to the number of transitions with the same labels or size of the problem as the rest of approaches. Fig.18 (b) illustrates this fact, see ML_3, ML_4 where the former model contains two duplicate transitions, and the later model has six duplicate transitions.

While the plots of Fig. 18 show the superiority of ILPSDP in terms of the execution time, in order to have more confidence on this trend we establish a *paired t-test* between the set of execution times by ILPSDP, and the one provided by A^*. The paired t-test is a statistical test where each subject or entity is measured twice, resulting in pairs of observations. In our setting, we executed each technique on every single dataset. In more detail, we propose the following hypotheses:

$$\begin{cases} H_0 : \text{True difference in average of execution times of ILPSDP and } A^* \text{ is equal to } 0 \\ H_1 : \text{True difference in average of execution times of ILPSDP and } A^* \text{ is not equal to } 0 \end{cases}$$

The result of this test according to numbers in Fig. 18, is presented in Table 5 as follows:

Table 5. Paired t-test between the execution times of the proposed approach and state of the art

Paired t-test					
Variables	D.F.	t statistic	p-value	95% Confidence Interval	Mean of diff.
ILPSDP's time, A^*'s time	17	4.4459	0.0001	[10122.06, 27472.01]	18797.04

In short, one can see that the p-value is 0.0001349; thus we can strongly reject H_0, and say that the execution times between ILPSDP and A^* are statistically different. It is worth noting that the reported 95% confidence interval provided by the t-test, i.e., [10122.06, 27472.01], has positive lower and upper bounds according to the test's setting, and A^* with 95% confidence has longer execution time than *ILPSDP* if they run on same datasets.

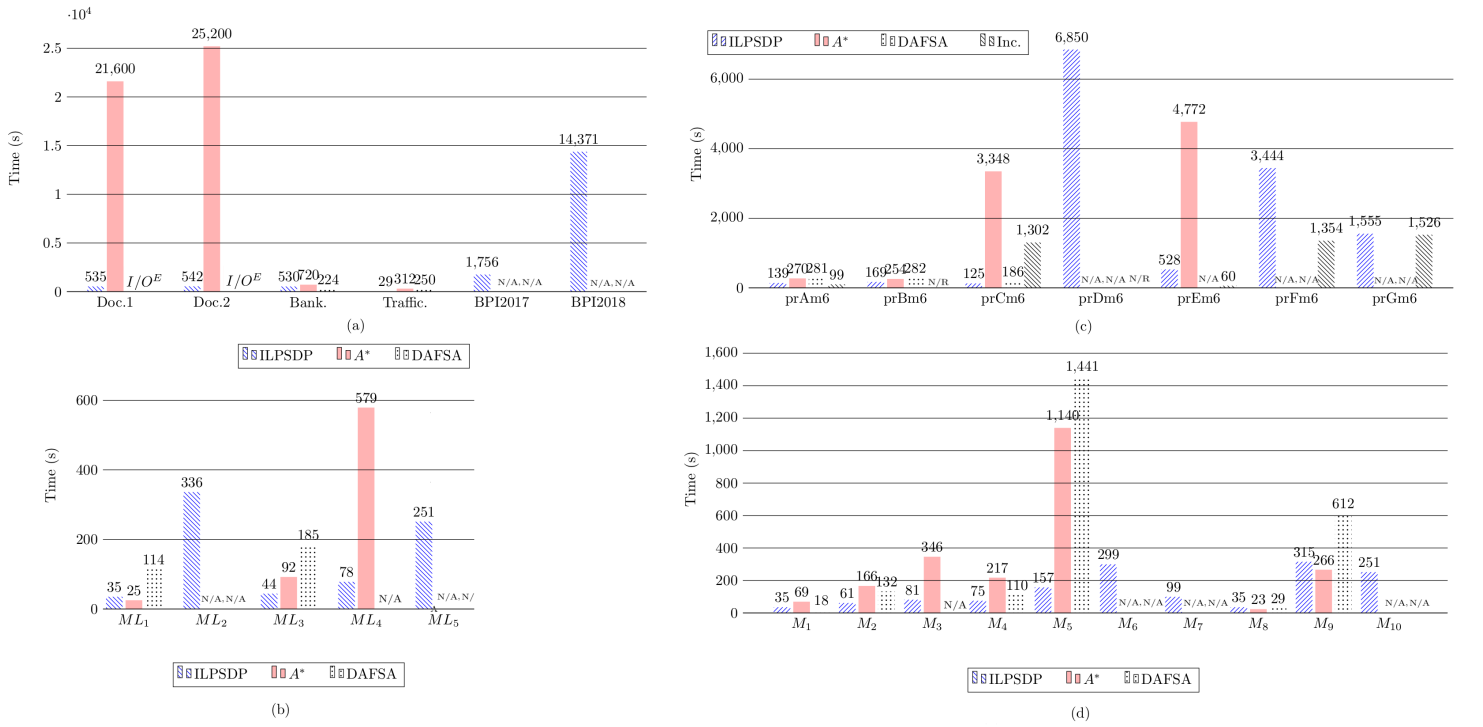


Fig. 18. (a) Realistic dataset [Taymouri and Carmona 2016b], [4dt 2018], (b) Synthetic datasets with duplicate labels, (c) Synthetic dataset [Munoz-Gama et al. 2014], (d) Synthetic dataset [Taymouri and Carmona 2016a]

Fitness Comparison. Figure 19 represents the distributions of fitness values, according to Def. 3.8, with $\delta_S = 1, \delta_L = 2$ and $\delta_M = 2$ for the proposed approach and A^*, across various datasets. Indeed, Fig. 19 contains box-plots of corresponding fitness values. These plots shed light on the performance of the proposed approach in terms of the quality of the solutions. One can see that the proposed approach, as mentioned in the paper, is not guaranteed to provide the optimal solution, however in many cases the average of fitness values are close to the optimal solutions, see $prAm6, prBm6, M_4, ML_1, ML_2$.

Though, the proposed approach provides in average close to optimal solutions, the variations in the provided solutions is greater than A^*; this can be seen by larger interquartile ranges of fitness values for the proposed technique in Fig. 19. In order to have statistical confidence about this metric between the two approaches, we establish a statistical test, i.e., paired t-test, for datasets that both approach provided results. In short, we consider the following hypotheses:

$$\begin{cases} H_0 : \text{True difference in means of fitness values of ILPSDP and } A^* \text{ is less than equal 0} \\ H_1 : \text{True difference in means of fitness values of ILPSDP and } A^* \text{ is greater than 0} \end{cases}$$

The result of this test according to numbers in Fig. 19, is presented in Table 6 as follows:

Table 6. Paired t-test between the fitness values of the proposed approach and state of the art

Paired t-test					
Variables	D.F.	t statistic	p-value	95% Confidence Interval	Mean of diff.
ILPSDP's fitness, A^*'s fitness	17	t = 5.2669	3.147e-05	[0.05618, inf]	0.08388

The p-value = $3.147e-05$ shows that A^* is better than ILPSDP in quality of alignments, i.e., fitness values. However, this is not a surprising fact since ILPSDP is based on heuristics, and there is no guarantee to find optimal solutions. Above that, the result of the paired t-test shows that alignment's fitness values provided by A^*, on average around 0.08, are higher than the corresponding values provided by ILPSDP. However, it is worth mentioning that the largest difference between fitness values found in our experiments is around 0.2.

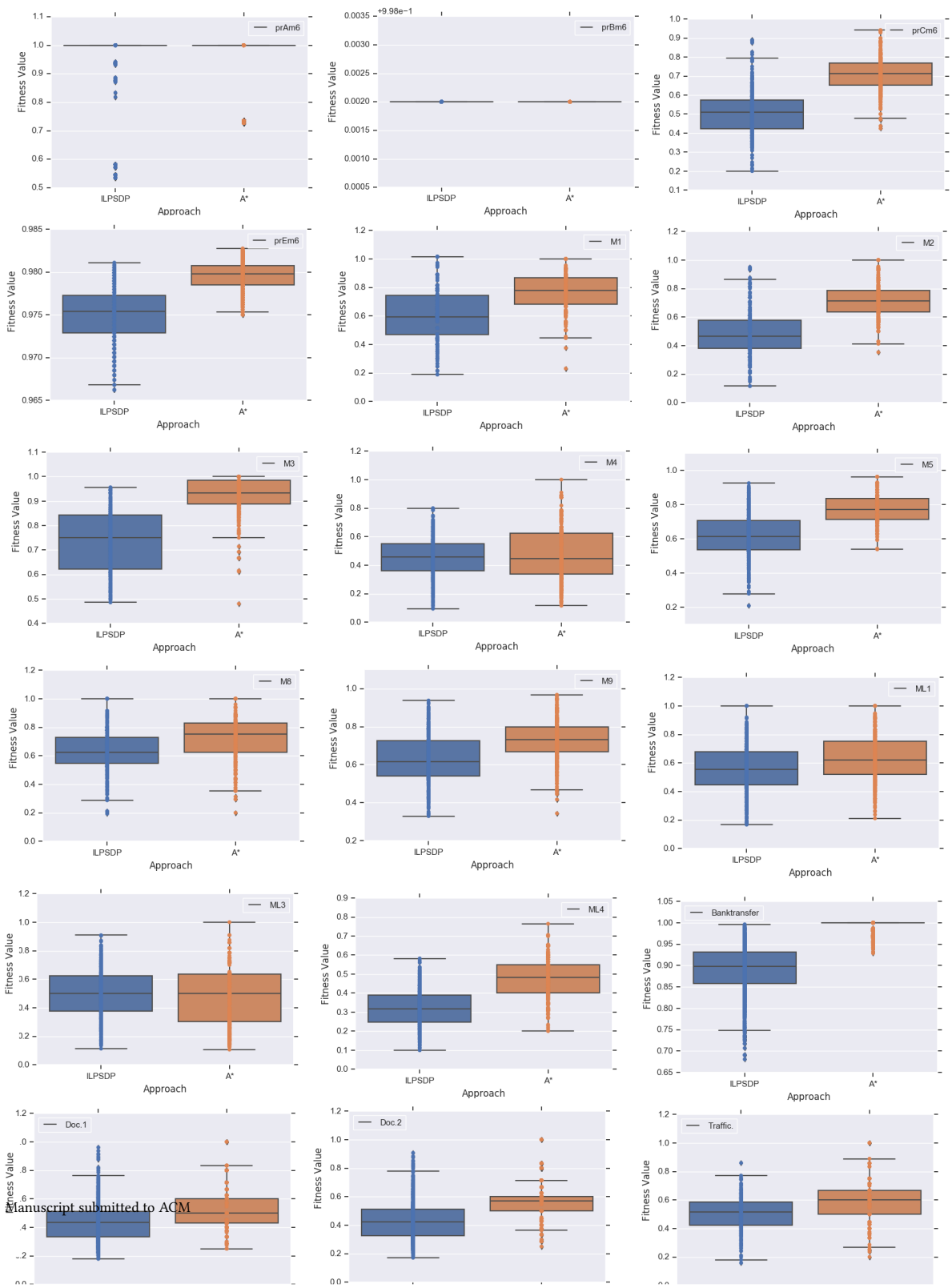


Fig. 19. Fitness distributions for the presented approach and the approach in [Adriansyah 2014]

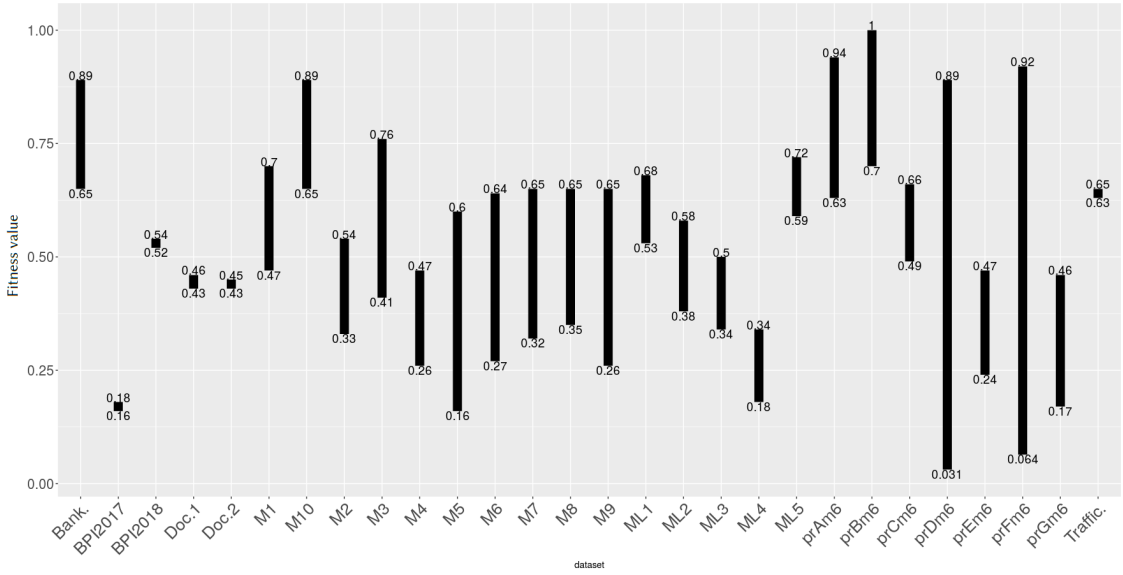


Fig. 20. Fitness values improvement

Improvement of Fitness Values. Fig. 20 reports the average of initial and final fitness values for the computed alignment, before and after reordering based on Def. 4.10. The range of a fitness value represents the initial value for the base alignment α_b up to a stable alignment where no fitness improvement can be obtained. One can see that for large models and observed traces there is a magnificent jump for the corresponding fitness values, and clearly models with massive parallelizations can get maximum benefits of the proposed technique, see *prDm6, prEm6* and *prFm6*.

Size Impact. To quantify how the presented approach is sensitive to the size of a given problem, i.e., in terms of the number of transitions of the model, we establish a linear regression between the number of transitions (T) and the corresponding execution time (time) in seconds for computing alignments given a process model. The result of the regresion for ILPSDP is:

$$\begin{cases} \text{time} = -918.01 + 11.13 * T \\ \text{F-statistic: } 14.63 \text{ on } 1 \text{ and } 26 \text{ DF, p-value: } 0.0007372 \end{cases} \tag{6}$$

and the regresion for A^* is:

$$\begin{cases} \text{time} = 8144.83 + 62.74 * T \\ \text{F-statistic: } 5.959 \text{ on } 1 \text{ and } 26 \text{ DF, p-value: } 0.02176 \end{cases} \tag{7}$$

First of all, according to the reported p-values of F-tests, one can see that both regression models are statistically significant at 5% confidence level. In other words, it shows that a linear relationship, statistically, can be used to model the association between the dependent variable (T) and the target variable (time). However, more sophisticated relations like exponential or nonlinear can be used to capture the existing association better. The above results state that the model's size has much more effect (around six times) on the A^* than on ILPSDP, see the corresponding slopes. Another interpretation of the above regression models is that, adding one extra transition to the model size, on average, increases the execution time around 62.74 unit of time (second) for A^* and 11.13 for *ILPSDP*.

Memory Consumption. The memory usage of ILPSDP, A^*, and DAFSA for all the benchmark datasets are represented in Fig.21.

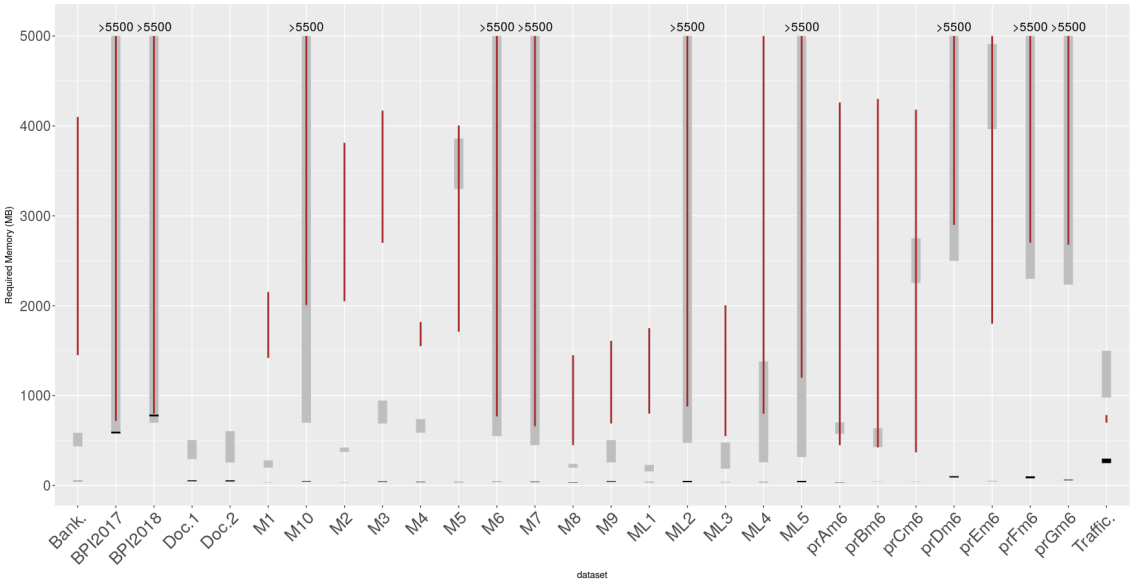


Fig. 21. Memory usage of the proposed approach (black), [Adriansyah 2014] (gray) and [Reißner et al. 2017] (red)

One can see that ILPSDP requires considerable less memory than the other two techniques for computing an alignment. For small and medium models, like *prAm6, prBm6, prCm6, Banktransfer*, the required memory for ILPSDP is at least 10 times less than the other approaches, and this ratio increases for large models with long traces. As an example, for *prDm6* ILPSDP required around 105MB whereas A^* needs more than 5500MB[11]. Notice that the memory consumption of ILPSDP is not sensitive to size of the model and length of the observed trace, whereas A^* is significantly sensitive, due to expanding and exploring the corresponding search spaces, see prDm6, prFm6, prGm6, M_6, M_7 and M_{10}. In addition,, the required memory for ILPSDP is not sensitive to the labels of transitions i.e., silent or duplicate labels, see $ML_1, \ldots, ML_5$.

6 CONCLUSION AND FUTURE WORK

Conformance checking is a crucial issue in diagnosing deviations of process models with their real behaviors. The best way to detect such deviations is by aligning observed and modeled behavior. A novel light and fast technique based on reordering of the initial alignment, which comes from replaying the resolution of an ILP instance is proposed in this paper. The technique has been implemented into a publicly available tool, and the evaluation shows promising capabilities to handle large instances including loops, silent transitions and transitions with duplicate labels. The evaluation reveals that this approach has very good performance in different perspectives, namely, execution time and memory consumption, and is able deal with large and as well as Spaghetti and well-structured models. Above that, experiments witnessed the obtained alignments are close to optimal alignments.

[11] For each of benchmark datasets prDm6, prFm6, prGm6, M_6, M_7, M_{10}, ML_2 and ML_5 the required memory for A^* is more than 5500MB, but due to limited amount of memory of the machine by which the experiments were done, the total required could not be measured.

As future work, we see many possibilities. First, we want to incorporate a mechanism to slide the observed trace during the reordering of an initial alignment α_b to get a better result. Second the mentioned replaying technique can be improved and being smarter whenever it considers next transitions ahead of the current transition being fired, which might result in a better initial solution. Third, we would like to improve the reordering technique so that merging multiple moves simultaneously is allowed. Finally an improvement in our implementation can be obtained based on the following fact: since for many cases the initial modeled trace is the same hence, it is not necessary to compute it for each case separately. Therefore by grouping observed traces with identical model trace we can have a huge reduction in resource usage.

Acknowledgments We would like to thank B. van Dongen for interesting discussions. This work has been supported by MINECO and FEDER funds under grant TIN2017-86727-C2-1-R.

REFERENCES

2018. 4TU: Centre for Research Data. http://researchdata.4tu.nl/home/

Arya Adriansyah. 2014. *Aligning observed and modeled behavior.* Ph.D. Dissertation. Technische Universiteit Eindhoven.

Andrea Burattin. 2016. PLG2: Multiperspective Process Randomization with Online and Offline Simulations. In *Proceedings of the BPM Demo Track 2016 Co-located with the 14th International Conference on Business Process Management (BPM 2016), Rio de Janeiro, Brazil, September 21, 2016.* 1–6. http://ceur-ws.org/Vol-1789/bpm-demo-2016-paper1.pdf

Josep Carmona, Boudewijn F. van Dongen, Andreas Solti, and Matthias Weidlich. 2018. *Conformance Checking - Relating Processes and Models.* Springer. https://doi.org/10.1007/978-3-319-99414-7

Massimiliano de Leoni and Andrea Marrella. 2017. Aligning Real Process Executions and Prescriptive Process Models through Automated Planning. *Expert Syst. Appl.* 82 (2017), 162–183.

J. Desel and J. Esparza. 1993. Reachability in cyclic extended free-choice systems. *TCS 114, Elsevier Science Publishers B.V.* (1993).

J. Desel and J. Esparza. 1995. *Free Choice Petri Nets.* Cambridge University Press, Cambridge, Great Britain.

J. Esparza and S. Melzer. 2000. Verification of safety properties using integer programming: Beyond the state equation. *Formal Methods in System Design* 16 (2000), 159–189.

Luciano García-Bañuelos, Nick R.T.P. van Beest, Marlon Dumas, Marcello La Rosa, and Willem Mertens. 2018. Complete and interpretable conformance checking of business processes. *IEEE Transactions on Software Engineering* 44, 3 (March 2018), 262–290. https://doi.org/10.1109/TSE.2017.2668418

Inc. Gurobi Optimization. 2016. Gurobi Optimizer Reference Manual. http://www.gurobi.com

R. P. Jagadeesh Chandra Bose and Wil van der Aalst. 2010. *Trace Alignment in Process Mining: Opportunities for Process Diagnostics.* Springer Berlin Heidelberg, Berlin, Heidelberg, 227–242. https://doi.org/10.1007/978-3-642-15618-2_17

Kristian Bisgaard Lassen and Wil M. P. van der Aalst. 2009. Complexity Metrics for Workflow Nets. *Inf. Softw. Technol.* 51, 3 (March 2009), 610–626. https://doi.org/10.1016/j.infsof.2008.08.005

Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. 2018. Scalable process discovery and conformance checking. *Software and System Modeling* 17, 2 (2018), 599–631.

Jorge Munoz-Gama, Josep Carmona, and Wil M. P. Van Der Aalst. 2014. Single-Entry Single-Exit Decomposed Conformance Checking. *Inf. Syst.* 46 (Dec. 2014), 102–122. https://doi.org/10.1016/j.is.2014.04.003

T. Murata. 1989. Petri Nets: Properties, Analysis and Applications. *Proc. IEEE* 77, 4 (April 1989), 541–574.

Richard Neapolitan. 2014. *Foundations Of Algorithms* (5th ed.). Jones and Bartlett Publishers, Inc., USA, 138–146.

Saul B. Needleman and Christian D. Wunsch. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* 48, 3 (1970), 443 – 453. https://doi.org/10.1016/0022-2836(70)90057-4

Artem Polyvyanyy, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Moe Thandar Wynn. 2017. Impact-Driven Process Model Repair. *ACM Trans. Softw. Eng. Methodol.* 25, 4 (2017), 28:1–28:60. https://doi.org/10.1145/2980764

Daniel Reißner, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, and Abel Armas-Cervantes. 2017. Scalable conformance checking of business processes. (March 2017). http://eprints.qut.edu.au/105118/ Paper submitted to "International Conference on Business Process Management (BMP 2017)" in Barcelona, Spain.

Anne Rozinat and Wil M. P. van der Aalst. 2008. Conformance checking of processes based on monitoring real behavior. *Inf. Syst.* 33, 1 (2008), 64–95. https://doi.org/10.1016/j.is.2007.07.001

M. Silva, E. Teruel, and J. M. Colom. 1998. Linear Algebraic and Linear Programming Techniques for the Analysis of Place/Transition Net Systems. In *Lecture Notes in Computer Science: Lectures on Petri Nets I: Basic Models*, Reisig, W. and Rozenberg, G. (Eds.). Vol. 1491. Springer-Verlag, 309–373.

Farbod Taymouri. 2017. ALI: Alignment for Large Instances. https://www.cs.upc.edu/~taymouri/tool.html

Farbod Taymouri and Josep Carmona. 2016a. Model and Event Log Reductions to Boost the Computation of Alignments. In *Proceedings of the 6th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2016), Graz, Austria, December 15-16, 2016.* 50–62. http://ceur-ws.org/Vol-1757/paper4.pdf

Farbod Taymouri and Josep Carmona. 2016b. A Recursive Paradigm for Aligning Observed Behavior of Large Structured Process Models. In *14th International Conference of Business Process Management (BPM), Rio de Janeiro, Brazil, September 18 - 22.*

Wil M. P. van der Aalst. 2013. Decomposing Petri nets for process mining: A generic approach. *Distributed and Parallel Databases* 31, 4 (2013), 471–507. https://doi.org/10.1007/s10619-013-7127-5

Wil M. P. van der Aalst. 2016. *Process Mining - Data Science in Action, Second Edition.* Springer. https://doi.org/10.1007/978-3-662-49851-4

Wil M. P. van der Aalst, Kees M. van Hee, Arthur H. M. ter Hofstede, Natalia Sidorova, H. M. W. Verbeek, Marc Voorhoeve, and Moe Thandar Wynn. 2011. Soundness of workflow nets: classification, decidability, and analysis. *Formal Asp. Comput.* 23, 3 (2011), 333–363. https://doi.org/10.1007/s00165-010-0161-4

Boudewijn van Dongen, Josep Carmona, Thomas Chatain, and Farbod Taymouri. 2017. Aligning Modeled and Observed Behavior: A Compromise Between Complexity and Quality. In *Proceedings of the 29th International Conference on Advanced Information Systems Engineering (CAiSE'17) (Lecture Notes in Computer Science)*, Eric Dubois and Klaus Pohl (Eds.), Vol. 10253. Springer, Essen, Germany. To appear.

Boudewijn F. van Dongen. 2018. Efficiently Computing Alignments - Using the Extended Marking Equation. In *Business Process Management - 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings.* 197–214. https://doi.org/10.1007/978-3-319-98648-7_12

B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. M. P. van der Aalst. 2005. The Prom Framework: A New Era in Process Mining Tool Support. In *Proceedings of the 26th International Conference on Applications and Theory of Petri Nets (ICATPN'05).* Springer-Verlag, Berlin, Heidelberg, 444–454. https://doi.org/10.1007/11494744_25

Seppe K. L. M. vanden Broucke, Jochen De Weerdt, Jan Vanthienen, and Bart Baesens. 2014. Determining Process Model Precision and Generalization with Weighted Artificial Negative Events. *IEEE Trans. Knowl. Data Eng.* 26, 8 (2014), 1877–1889. https://doi.org/10.1109/TKDE.2013.130

H. M. W. Verbeek and W. M. P. van der Aalst. 2016. *Merging Alignments for Decomposed Replay.* Springer International Publishing, Cham, 219–239. https://doi.org/10.1007/978-3-319-39086-4_14

A PROOFS

A.1 Proof of Lemma 4.4

Proof. The proof has two parts, the first part shows the existence of $\widehat{\sigma_O}$ and the second part establishes there is a sequence of firing whose Parikh vector is $\widehat{\sigma_O}$. The second part is crucial due to the fact that Parikh vectors in general are not guaranteed to be executable (see the example provided at the end of Sect. 3.2), i.e., the final marking is not reachable by firing the elements of the Parikh vector.

- *Part 1:*

  This part proceeds by contradiction. Consider a transition like $t_j \in T_{m_i} \cap supp(\widehat{\sigma_P})$ with $\widehat{\sigma_P}[t_j] = n_j$. To proceed, separate situations are considered as follows:

  (1) Let $\ell(t_j) = a_j \in supp(\widehat{\sigma})$ with $\widehat{\sigma}[a_j] = n$, in other words for transition t_j the corresponding events, i.e., a_j, happened n times in the observed trace σ. Let's assume that it is able to fire only n_i times, with $n_i < n_j$, and there is no Parikh vector to fire it n_j times. In other words there exist no $\widehat{\sigma_O}$ as stated by the lemma. However this is impossible due to violation of the following constraint of Eq. (4):

$$\underbrace{\widehat{\sigma}[a_j]}_{n} = \underbrace{X[t_j]}_{n_j} + \underbrace{X^s[t_j]}_{n_2} + \underbrace{Res}_{n_3} \quad [12] \quad \text{where} \quad n = n_j + n_2 + n_3$$

  Stated differently, n_i times firing of t_j makes the equality sign " = " to become " > " since the difference value $n_j - n_i$ cannot be compensated by other terms in that equation. Therefore there is a Parikh vector $\widehat{\sigma_O}$ in which t_j fires n_j times.

[12] In case of multiple transitions with the same label, for the sake of simplicity and comprehension the other corresponding terms are represented by the residual term, i.e., Res.

(2) Let $\ell(t_j) = a_j \notin supp(\widehat{\sigma})$, in other words t_j is a skipped transition. Now suppose that it fires only n_i times, with $n_i < n_j$, i.e., there is no Parikh vector like $\widehat{\sigma_O}$ to fire it n_j times, and for the next markings all the enabled transitions are fired without violating any constraints with the absence of some t_j. However, this is impossible because in that case at the end we reach a solution like $\widehat{\sigma'_P}$ with $\widehat{\sigma'_P}[t_j] < \widehat{\sigma_P}[t_j]$ such that the objective function related to $\widehat{\sigma'_P}$ is greater than its counterpart, which corresponds to $\widehat{\sigma_P}$ whereas the later is supposed to be the maximum one based on given constraints in Eq. (4). Therefore there is a Parikh vector $\widehat{\sigma_O}$ in which t_j fires n_j times.

- *Part 2:*

  Suppose that $\widehat{\sigma_P}[t_j] = n_j > 1$, this means that t_j might be in a loop and the aim is to show that there exist a firing sequence in which t_j fires n_j times.

  (1) Let's first assume that it is in a loop. In this case given the system net, i.e., SN, one can easily form a cyclic allocation α according to Def. 3.2 with a non-empty domain C which contains t_j. Then based on *Cyclic Allocation Lemma* presented in [Desel and Esparza 1995] for live and bounded FC-models[13] there exist an occurrence sequence $m_{start} \xrightarrow{x\delta}$ such that:
  – x is a finite and contains no transitions of C
  – δ is infinite and contains only transitions allocated by α
  The mentioned Lemma states that there exist a firing sequence in which t_j fires n_j times.

  (2) Let's assume that t_j is not in a loop, then there should be enough tokens in ${}^\bullet t_j$ such that if fires n_j times since otherwise it violates the soundness property of the given model.

□

A.2 Proof of Theorem 4.3

Based on Lemmas 4.4 and 4.5, the proof of Theorem 4.3 is as follow:

Proof. It is assumed that the model is deadlock free (except the final marking $m_{end} = \{p_e\}$). We proceed by contradiction, namely, legitimate firing of a transition causes to miss at least one of its enabled siblings, therefore, backtracking must be achieved. Stated differently, given the context provided by Def. 4.2, to fire t_j after t_k, backtracking needs to be done. However this is impossible due to the following argument. Firing t_k consumes tokens from ${}^\bullet t_k =^\bullet t_j$ that might make it unmarked which in that case based on Lemma 4.5 there is a set of transitions like T_M which marks ${}^\bullet t_j$. On the other hand, based on the firing policy in Def. 4.1 all the elements in T_M are at most as close as t_j to the final marking since otherwise based on Def. 4.1 they are fired earlier. Hence there is an arc-back path from T_M to elements of ${}^\bullet t_j$. By the deadlock free assumption of the model this path amounts to $D(t_k, t_j)$, thus if $D(t_k, t_j) \neq \infty$ then the places in ${}^\bullet t_j$ are marked and therefore t_j is fired without backtracking. □

A.3 Proof of Theorem 4.6

Proof. Since SN is a WF-net, let ${}^\bullet m_{end} = \{t_{end}\}$ and $m^\bullet_{start} = \{t_{start}\}$, stated differently t_{start} and t_{end} are the only transitions of the model which consumes token from the initial marking and puts token to the final marking respectively. The first part of proof proceeds by contradiction. Assume that there is no σ_R by which m_{end} is reachable. Since t_{end} is the only one element which marks m_{end} this assumption indicates $t_{end} \notin supp(\widehat{\sigma_R})$ which implies

[13] We can easily make SN live by connecting the final place to the start place by a silent transition τ, a fact that does not harm the proof since the lemma holds for the elements different from τ.

$t_{end} \notin supp(\widehat{\sigma_P})$. However this is impossible since in that case the solution of Eq. (4) is infeasible and there is no solution. Thus t_{end} will be fired. By the same token t_{start} will be fired because it is the only one transition of the model which consumes the initial token. Put it differently $t_{start}, t_{end} \in supp(\widehat{\sigma_R})$. The second part of proof continues as follow. Due to firing of t_{end}, some elements of $\widehat{\sigma_P}$, denoted by $\widehat{\sigma_{P_k}}$[14] must be fired to mark ${}^\bullet t_{end}$ since otherwise some tokens will be missed and therefore it contradicts the solution of Eq. (4). Also, there is the same argument for elements of $\widehat{\sigma_{P_k}}$ as well, and it continues until we reach transition(s) of the model which consume(s) the produced token(s) by t_{start}. More formally:

$$
\begin{aligned}
m_{end} = m_k + \mathbf{N}X_{k+1}, \qquad & X_{k+1}[t_{end}] = 1, \quad \text{o/w} \quad 0 \quad \text{and} \quad m_k \geq 0 \\
m_k = m_{k-1} + \mathbf{N}X_k, \qquad & \forall t \in \widehat{\sigma_{P_k}}, X_k[t] = 1, \quad \text{o/w} \quad 0 \quad \text{and} \quad m_{k-1} \geq 0 \\
& \vdots \\
m_2 = m_1 + \mathbf{N}X_2, \qquad & \forall t \in \widehat{\sigma_{P_2}}, X_2[t] = 1, \quad \text{o/w} \quad 0 \quad \text{and} \quad m_1 \geq 0 \\
m_1 = m_{start} + \mathbf{N}X_1, \qquad & X_1[t_{start}] = 1, \quad \text{o/w} \quad 0 \quad \text{and} \quad m_{start} \geq 0
\end{aligned}
$$

This argumentation establishes the existence of a sequence like σ_R by which the final marking m_{end} is reachable from m_{start} and $\widehat{\sigma_R} \leqq \widehat{\sigma_P}$. □

A.4 Proof of Theorem 4.7

Proof. The proof proceeds by contradiction. Suppose that for the proposed approach $\nexists \sigma_N$ such that $m_{start}[\sigma_N\rangle m_{end}$. As such, it means the proposed technique reaches to an arbitrary marking $m_i \neq m_{end}$ for which there are no enabled transitions, i.e., $T_{m_i} = \phi$. However this is impossible due to the following reasons:

(1) There is no deadlock in the model (except the final marking).
(2) By virtue of Theorem 4.6 $\exists \sigma_R$ such that $m_{start}[\sigma_R\rangle m_{end}$ therefore the initial assumption made at the beginning of the proof implies backtracking since $m_i \geq 0$ and $m_i \neq m_{end}$ but it contradicts the presented replaying technique since by virtue of Theorem 4.3 it does not backtrack.

Therefore the presented technique finds a sequence like σ_N such that $m_{start}[\sigma_N\rangle m_{end}$. □

A.5 Proof of Theorem 4.8

Proof. We present the proof by contradiction as follows. Suppose that $\exists \sigma'_N$ where $\widehat{\sigma_N} \leqq \widehat{\sigma'_N} \leqq \widehat{\sigma_P}$ and $|\sigma_N| < |\sigma'_N|$ such that $m_{start}[\sigma'_N\rangle m_{end}$, in other words $\exists t_i \in supp(\widehat{\sigma'_N})$ but $t_i \notin supp(\widehat{\sigma_N})$. The mentioned assumption implies that there is marking m'_i such that ${}^\bullet t_i \leq m'_i$ and it is not met while σ_N is being obtained. But this is impossible to have such an intermediate marking for the following reason. If we consider the corresponding reachability markings for two sequences, i.e., σ_N and σ'_N, as follows:

$$
\begin{aligned}
& m_{start} \xrightarrow{t_{start}} \cdots m_i \cdots \xrightarrow{t_{end}} m_{end}, && \sigma_N \\
& m_{start} \xrightarrow{t_{start}} \cdots m_i \xrightarrow{t_j} m'_i \xrightarrow{t_i} m'_{i+1} \cdots \xrightarrow{t_{end}} m_{end}, && \sigma'_N
\end{aligned}
$$

Then to reach m'_i some transitions like t_j[15] from the previous marking, let's say m_i, must be fired. If m_i is reachable while σ_N is being replied then m'_i is also reachable by virtue of Theorem. 4.3, since all the enabled transitions of m_i will

[14] If $supp(\widehat{\sigma_{P_k}}) > 1$ then these elements are fired concurrently, i.e., there is no causality among them.
[15] To make everything simple assumes only one transition is enough. It can be easily rephrased for the case with many transitions.

be fired. Indeed it is impossible to have such a marking like m'_i and corresponding previous markings like m_i which are not met while σ_N is replayed since the initial marking of both σ_N and σ'_N is m_{start}.

□